Contents

1 Introduction

2 Naming Conventions

3 Model Architecture

4 Model Configuration Options

5 Simulink

6 Stateflow

A Recommendations for Automation Tools

B Guideline Writing

C Flowchart Reference

D Background Information on Basic Blocks and Signals

MAAB Glossary

1

Introduction

Presentation of Guidelines Hosted by MathWorks

This presentation of the MathWorks Automotive Advisory Board (MAAB) guidelines, Version 2.1, is based on the document, of the same title, authored by the MAAB working group. In addition to the information included in the original document, this presentation includes references to corresponding Model Advisor MAAB checks that you can apply if you are licensed to use Simulink® and Simulink® Verification and Validation™ software.

Motivation

The MathWorks Automotive Advisory Board (MAAB) guidelines are important for project success and teamwork—both in-house and when cooperating with partners or subcontractors. Observing the guidelines is a key prerequisite to achieving:

- Problem-free system integration
- Well-defined interfaces
- Uniform appearance of models, code, and documentation
- Reusable models
- Readable models
- Problem-free exchange of models
- A simple, effective process
- Professional documentation
- Understandable presentations
- Fast software changes
- Cooperation with subcontractors
- Successful transitions of research or predevelopment projects to product development

Guideline Template

In this section...

Guideline descriptions are documented, using the following template. Companies that want to create additional guidelines are encouraged to use the same template.

ID: Title	*XX_nnnn*: Title of the guideline (unique, short)
Priority	Mandatory, Strongly recommended, or Recommended
Scope	MAAB, NA-MAAB, J-MAAB, Specific Company (for optional local company usage)
MATLAB® Versions	One of the following: All RX, RY, RZ RX and earlier RX and later RX through RY
Prerequisites	Links to guidelines, which are prerequisites to this guideline (ID: Title)

Description	Description of the guideline (text, images)
Rationale	Motivation for the guideline
Last Change	Version number of last change
Model Advisor Check	Title of and link to the corresponding Model Advisor check, if a check exists

Note The elements of this template are the minimum required items for understanding and exchanging guidelines. You can add project or vendor fields to this template as long as their meaning does not overlap with existing fields. Such additions are encouraged if they help to integrate other guideline templates and lead to a wider acceptance of the core template.

Guideline ID

- The guideline ID is built out of two lowercase letters (representing the origin of the rule) and a four-digit number, separated by an underscore.
- Once a new guideline has an ID, the ID does not change.
- The ID is used for references to guidelines.
- The two letter prefixes **na**, **jp**, **jc** and **eu** are reserved for future MAAB committee rules.
- Legacy prefixes, **db**, **jm**, **hd**, and **ar**, are reserved. The MAAB committee will not use these prefixes for new rules.
- No new rules are to be written with these legacy prefixes.

Guideline Title

- The title should be a short, but unique description of the guidelines area of application (for example, length of names)
- The title is used for the Prerequisites field and for custom checker tools.
- The title text should appear with a hyperlink that links to the guideline.

Note The title should not be a redundant short description of the guidelines content, because while the latter may change over time, the title should remain stable.

Priority

Each guideline must be rated with one of the following priorities:

- Mandatory
- Strongly recommended
- Recommended

The priority describes the importance of the guideline and determines the consequences of violations.

Mandatory	Strongly Recommended	Recommended
Definition		
Guidelines that all companies agree to that are absolutely essential Guidelines that all companies conform to 100%	Guidelines that are agreed upon to be a good practice, but legacy models preclude a company from conforming to the guideline 100% Models should conform to these guidelines to the greatest extent possible; however, 100% compliance is not required	Guidelines that are recommended to improve the appearance of the model diagram, but are not critical to running the model Guidelines where conformance is preferred, but not required
Consequences: If the guideline is violated,		

Mandatory	Strongly Recommended	Recommended
Essential items are missing The model might not work properly	The quality and appearance deteriorates An adverse effect on maintainability, portability, and reusability might occur	The appearance does not conform with other projects
Waiver Policy: If the guideline is intentionally ignored,		
The reasons must be documented		

Scope

The scope of a guideline may be set to one of the following:

Scope	Description
MAAB (MathWorks Automotive Advisory Board)	A group of automotive manufacturers and suppliers that work closely together with MathWorks. MAAB includes the subgroups J-MAAB and NA-MAAB.
J-MAAB (Japan MAAB)	A subgroup of MAAB that includes automotive manufacturers and suppliers in Japan and works closely with MathWorks. Rules with J-MAAB scope are local to Japan.
NA-MAAB (North American MAAB)	A subgroup of MAAB that includes automotive manufacturers and suppliers in the United States and Europe and works closely with MathWorks. Rules with NA-MAAB scope are local to the United States and Europe.

MATLAB Versions

The guidelines support all versions of the MATLAB and Simulink products. If the rule applies to specific versions, the versions are identified in the MATLAB versions field. The version information is in one of the following formats.

Format	Definition
All	All versions of MATLAB
RX, RY, or RZ	A specific version of MATLAB
RX and earlier	Versions of MATLAB until version RX
RX and later	Versions of MATLAB from version RX to the current version
RX through RY	Versions of MATLAB between RX and RY

Prerequisites

- The Prerequisite field is for links to other guidelines that are prerequisites for this guideline (logical conjunction).
- Use the guideline ID (for consistency) and the title (for readability) for the links.
- The Prerequisites field should not contain any other text.

Description

- This field contains a detailed description of the guideline.
- If needed, add images and tables.

Note If formal notation (math, regular expression, syntax diagrams, and exact numbers/limits) is available, use it to unambiguously describe a guideline and specify an automated check. However, a human, understandable, informal description must always be provided for daily reference.

Rationale

This field lists the reasons that apply for a given guideline. You can recommend guidelines for one or more of the following reasons:

Rationale	Description
Readability	Easily understood algorithms • Readable models • Uniform appearance of models, code, and documentation • Clean interfaces • Professional documentation
Workflow	Effective development process and workflow • Ease of maintenance • Rapid model changes • Reusable components • Problem-free exchange of models • Model portability
Simulation	Efficient simulation and analysis • Simulation speed • Simulation memory • Model instrumentation
Verification and validation	Ability to verify and validate a model and generated code with • Requirements traceability • Testing • Problem-free system integration • Clean interfaces
Code generation	Generation of code that is efficient and effective for embedded systems • Fast software changes • Robustness of generated code

Last Change

The Last change field contains the document version number.

Model Advisor Check

The Simulink Verification and Validation product includes Simulink Model Advisor MAAB checks, which correspond to a subset of MAAB guidelines, that you can select and run with the Simulink Model Advisor. In this presentation of the MAAB guidelines, MathWorks includes a Model Advisor check field in guideline descriptions, which contains the title of and a link to the corresponding Model Advisor check, if a check exists. Although this information is included, note that the MAAB working group takes a neutral stance on recommendations for style guide checkers.

For a list of available Model Advisor checks for the MAAB guidelines, see “MathWorks® Automotive Advisory Board Checks” in the Simulink Verification and Validation documentation. For information on using the Model Advisor, see “Consulting the Model Advisor” in the Simulink documentation.

Document Usage

- Chapter 2, “Naming Conventions” and Chapter 3, “Model Architecture” provide basic guidelines that apply to all types of models.
- Chapter 5, “Simulink” and Chapter 6, “Stateflow” deal with specific rules for those environments.
- Some guidelines are dependent on other guidelines and are explicitly listed throughout the document.

For information on automated checking of the guidelines, see Appendix A, “Recommendations for Automation Tools”.

2

Naming Conventions

General Guidelines

- `ar_0001: Filenames`
- `ar_0002: Directory names`

ID: Title	ar_0001: Filenames
Priority	Mandatory
Scope	MAAB
MATLAB Versions	All
Prerequisites	None

Description

A file name conforms to the following constraints:

Form

filename = *name*.*extension*

- *name*: no leading digits, no blanks
- *extension*: no blanks

Uniqueness

All file names within the parent project directory

Allowed Characters

name:
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 _

extension:
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9

Underscores

name:

- Can use underscores to separate parts
- Cannot have more than one consecutive underscore
- Cannot start with an underscore
- Cannot end with an underscore

extension:

Should not use underscores

Rationale

- Readability
- Workflow

Last Changed

V1.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Naming Conventions > "Check for incorrect file names"

ar_0002: Directory names

Priority Mandatory

Scope MAAB

MATLAB Versions All

Prerequisites None

Description A directory name conforms to the following constraints:

Form

directory name = *name*

name: no leading digits, no blanks

Uniqueness

All directory names within the parent project directory

Allowed characters

name:
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 _

Underscores

name:

- Can use underscores to separate parts
- Cannot have more than one consecutive underscore
- Cannot start with an underscore
- Cannot end with an underscore

Rationale

- Readability
- Workflow

ar_0002: Directory names

Last Changed	V1.0
Model Advisor Check	**By Task > Modeling Standards for MAAB > Naming Conventions > "Check folder names"**

Model Content

- jc_0201: Usable characters for Subsystem names
- jc_0211: Usable characters for Inport blocks and Outport blocks
- jc_0221: Usable characters for signal line names
- jc_0231: Usable characters for block names
- na_0014: Use of local language in Simulink and Stateflow

jc_0201: Usable characters for Subsystem names

ID: Title jc_0201: Usable characters for Subsystem

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description The names of all Subsystem blocks should conform to the following constraints:

Form

name:

- Should not start with a number
- Should not include blank spaces

Allowed Characters

name:
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 _

Underscores

name:

- Can use underscores to separate parts
- Cannot have more than one consecutive underscore
- Cannot start with an underscore
- Cannot end with an underscore

jc_0201: Usable characters for Subsystem names

Rationale

- Readability
- Workflow
- Code generation

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Naming Conventions > "Check subsystem names"

jc_0211: Usable characters for Inport blocks and Outport blocks

ID: Title	jc_0211: Usable characters for Inport blocks and Outport blocks
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None

Description

The names of all Inport blocks and Output blocks should conform to the following constraints:

Form

name:

- Should not start with a number
- Should not include blank spaces

Allowed Characters

name:
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 _

Underscores

name:

- Can use underscores to separate parts
- Cannot have more than one consecutive underscore
- Cannot start with an underscore
- Cannot end with an underscore

jc_0211: Usable characters for Inport blocks and Outport blocks

Rationale

- Readability
- Workflow
- Code generation

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Naming Conventions > "Check port block names"

jc_0221: Usable characters for signal line names

ID: Title jc_0221: Usable characters for signal line names

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description Identifies named signals constraints

Form

name:

- Should not start with a number
- Should not include blank spaces
- Should not include any control characters

Allowed Characters

name:
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 _

Underscores

name:

- Can use underscores to separate parts
- Cannot have more than one consecutive underscore
- Cannot start with an underscore
- Cannot end with an underscore

jc_0221: Usable characters for signal line names

Rationale

- Readability
- Workflow
- Code generation

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Naming Conventions > "Check signal labels for incorrect characters"

jc_0231: Usable characters for block names

ID: Title jc_0231: Usable characters for block names

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites jc_0201: Usable characters for Subsystem names

Description The names of all blocks should conform to the following constraints:

Form

name:

- Should not start with a number
- Should not include spaces at the beginning of a block name
- Should not use double byte characters
- Carriage returns are allowed

Allowed Characters

name:
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 _

Note This rule does not apply to Subsystem blocks.

Rationale

- Readability
- Workflow
- Code generation

jc_0231: Usable characters for block names

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Naming Conventions > "Check block names for incorrect characters"

na_0014: Use of local language in Simulink and Stateflow

ID: Title na_0014: Use of local language in Simulink and Stateflow®

Priority Strongly recommended

Scope J-MAAB

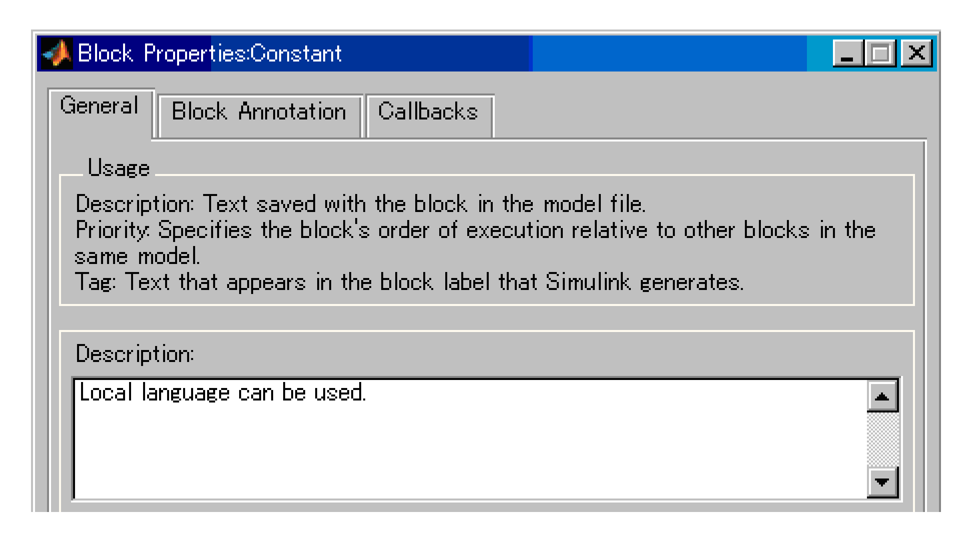
MATLAB Versions All

Prerequisites None

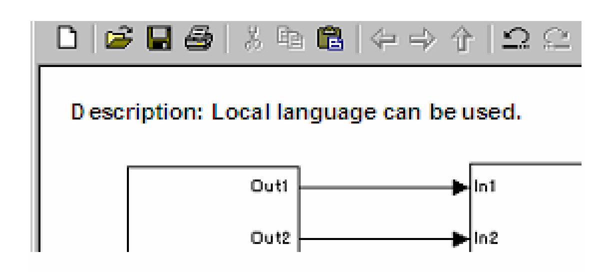
Description The local language should be used in descriptive fields only. Descriptive fields are text entry points that do not affect code generation or simulation. Examples of descriptive fields include the **Description** field in the Block Properties dialog box.

Simulink Examples

- The **Description** field in the Block Properties dialog box



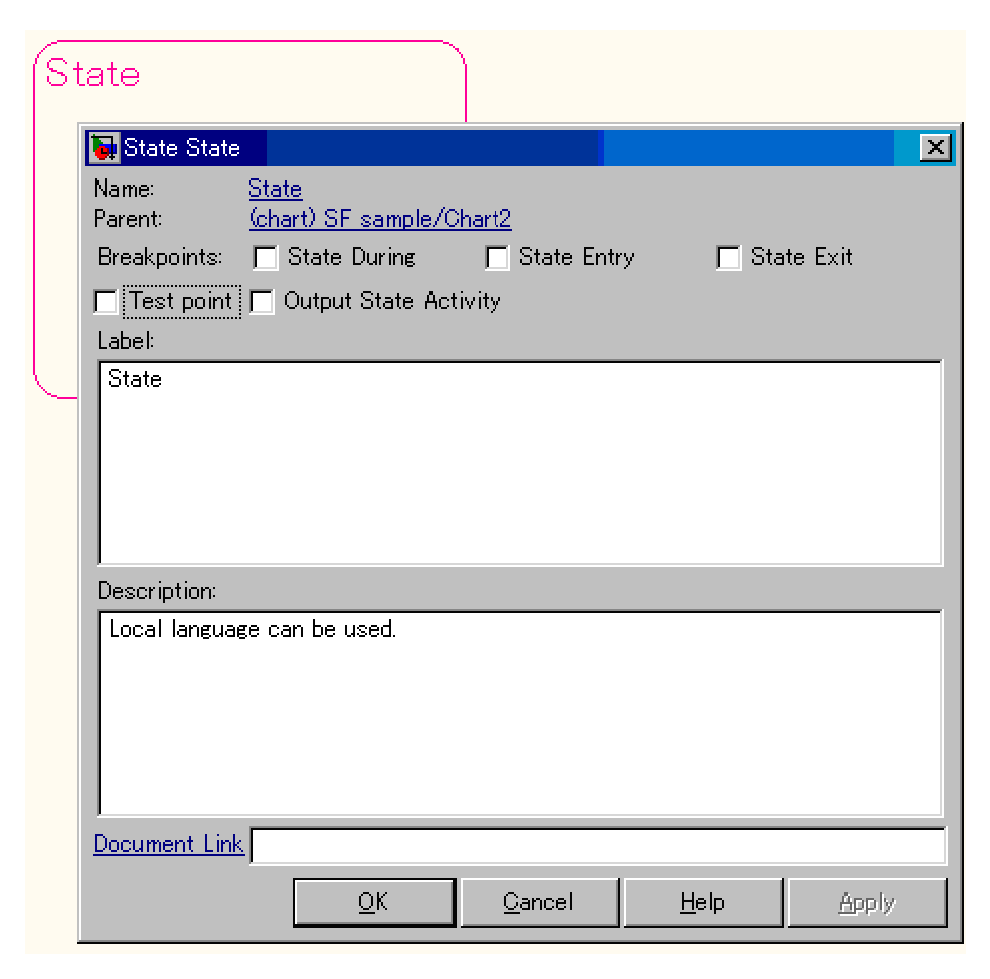
- Text annotation entered directly in the model



Stateflow Examples

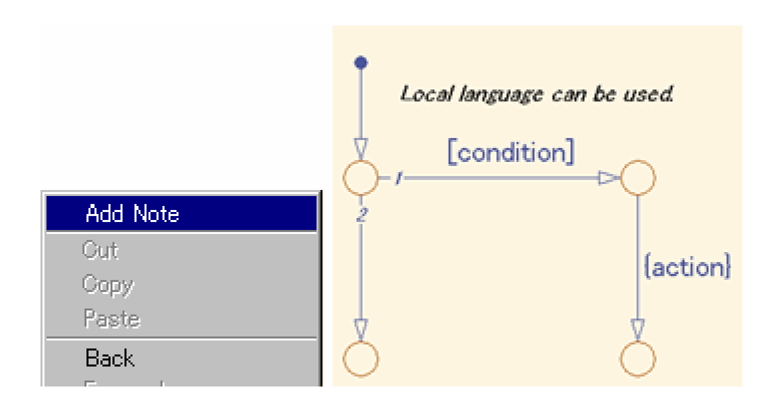
- The **Description** field of chart and state Properties

na_0014: Use of local language in Simulink and Stateflow



- Annotation description added using **Add Note**

na_0014: Use of local language in Simulink and Stateflow



Note It is possible that Simulink cannot open a model that includes local language on different character encoding systems. Therefore, pay attention when using local characters for exchanging models between countries.

Rationale

- Readability
- Workflow

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Naming Conventions > "Check signal labels for incorrect characters"

na_0014: Use of local language in Simulink and Stateflow

Model Architecture

This document uses the term *basic blocks* to refer to blocks built into the Simulink block libraries. "Basic Blocks" on page D-2 in Appendix D, "Background Information on Basic Blocks and Signals" lists some examples of basic blocks.

Simulink and Stateflow Partitioning

- `na_0006: Guidelines for mixed use of Simulink and Stateflow`
- `na_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines`

na_0006: Guidelines for mixed use of Simulink and Stateflow

ID: Title na_0006: Guidelines for mixed use of Simulink and Stateflow

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description The choice of whether to use Simulink or Stateflow to model a given portion of the control algorithm functionality should be driven by the nature of the behavior being modeled.

- If the function primarily involves complicated logical operations, use Stateflow diagrams.

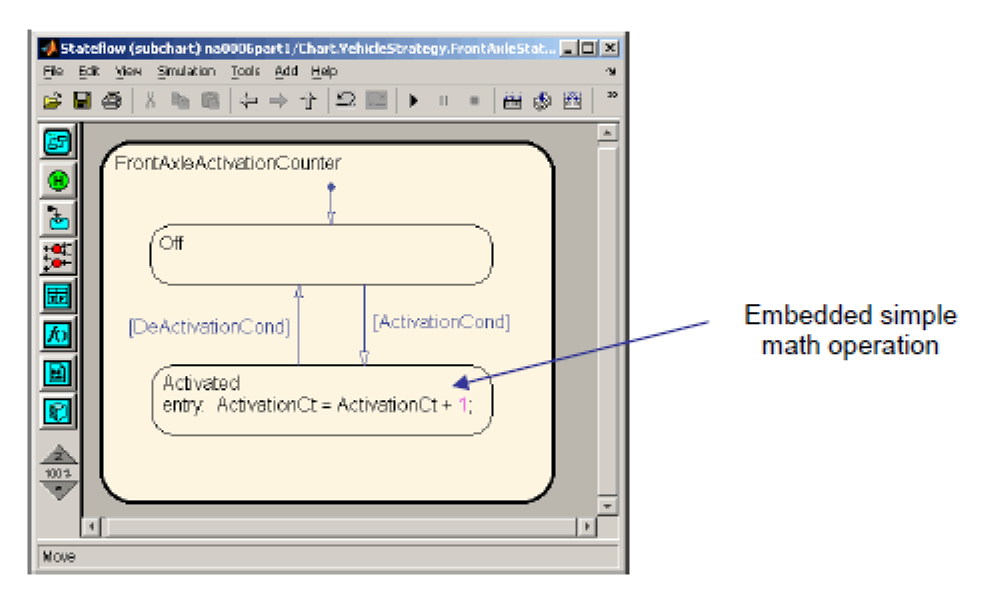
  Use Stateflow diagrams to implement modal logic, where the control function to be performed at the current time depends on a combination of *past and present logical conditions*.

- If the function primarily involves numerical operations, use Simulink features.

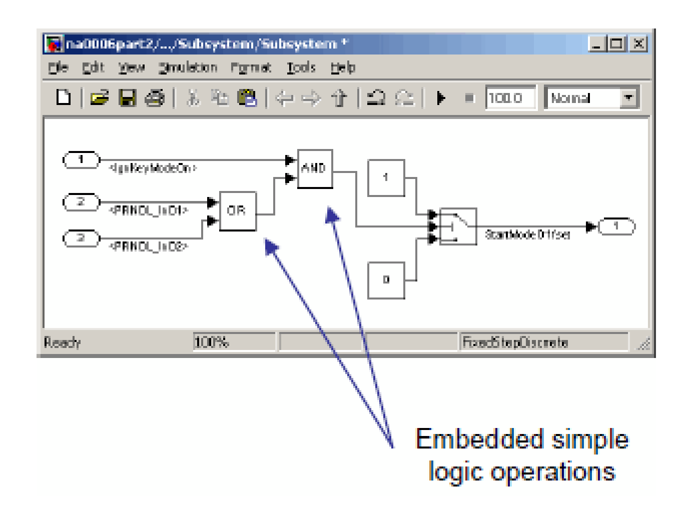
Specifics

- If the primary nature of the function is logical, but some simple numerical calculations are done to support the logic, implement the simple numerical functions using the Stateflow action language.

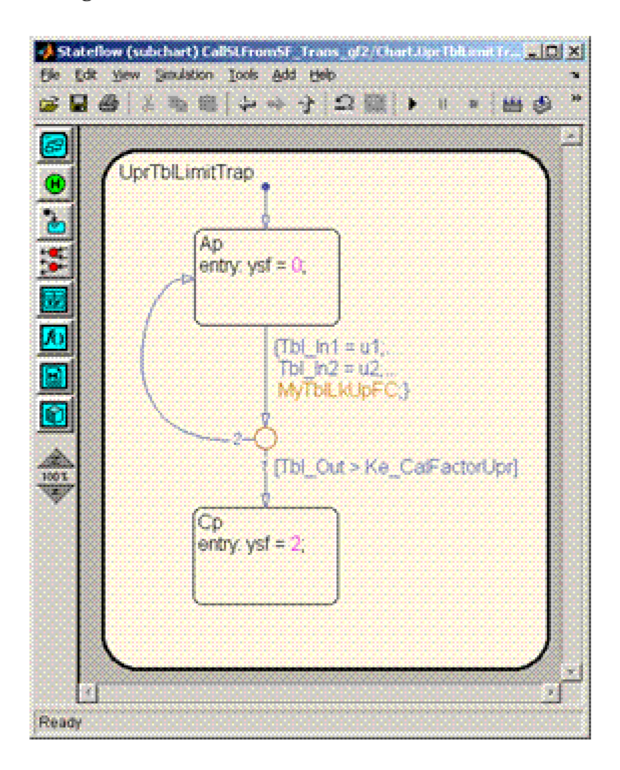
na_0006: Guidelines for mixed use of Simulink and Stateflow



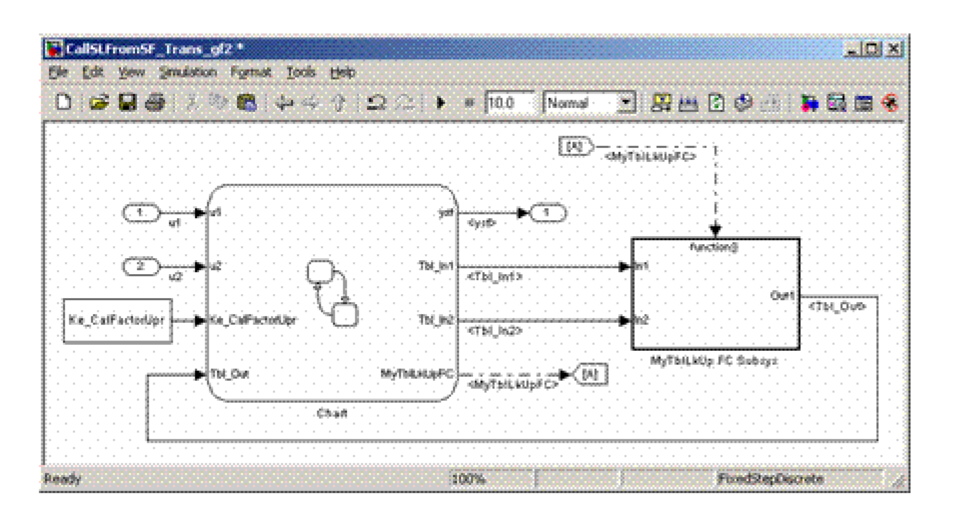
- If the primary nature of the function is numeric, but some simple logical operations are done to support the arithmetic, implement the simple logical functions with Simulink blocks.

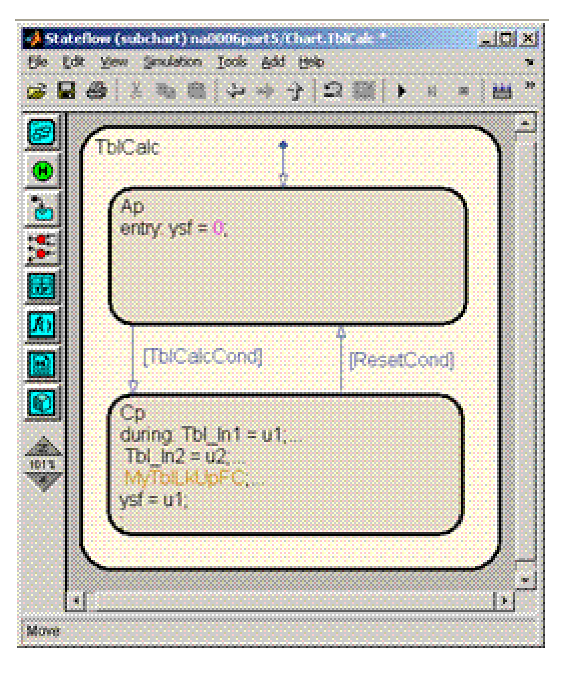


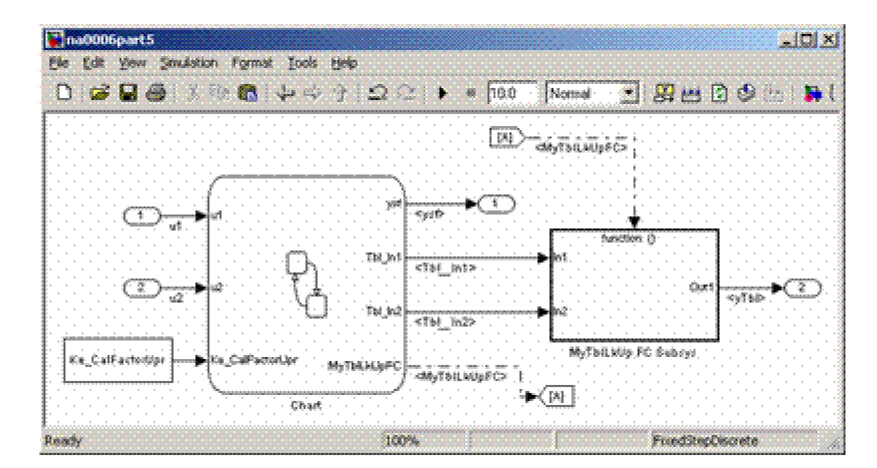
- If the primary nature of the function is logical, and some complicated numerical calculations must be done to support the logic, use a Simulink subsystem to implement the numerical calculations. The Stateflow software should invoke the execution of the subsystem, using a function call.



na_0006: Guidelines for mixed use of Simulink and Stateflow

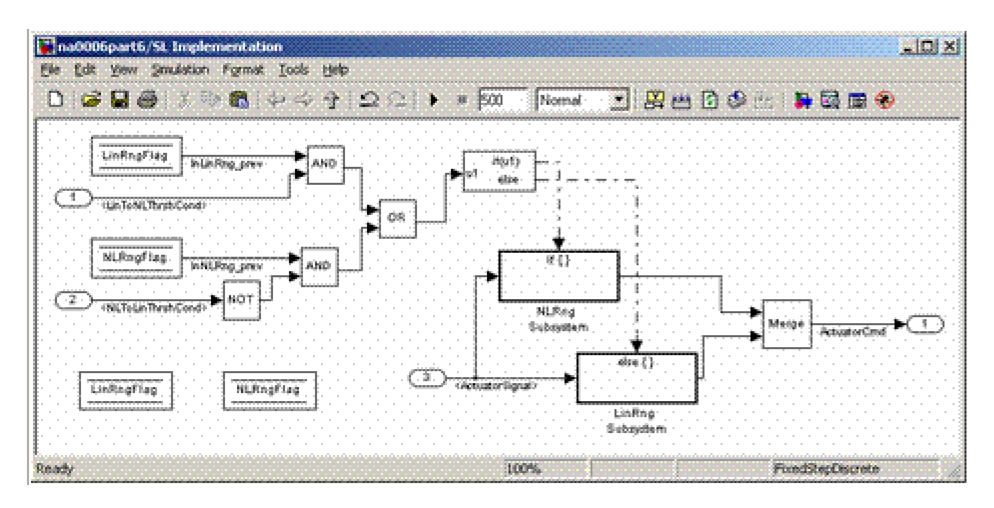


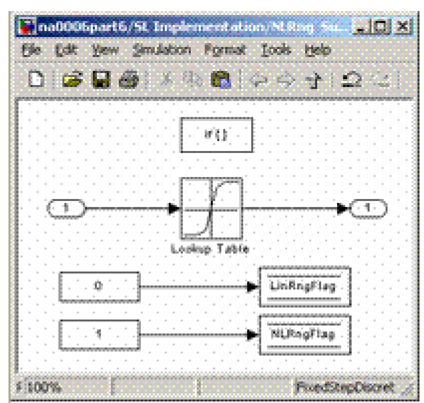




- Use the Stateflow product to implement modal logic, where the control function to be performed at the current time depends on a combination of *past and present logical conditions*. (If there is a need to store the result of a logical condition test in a Simulink model, for example, by storing a flag, this is an indicator of the presence of modal logic, which should be modeled with Stateflow software.)

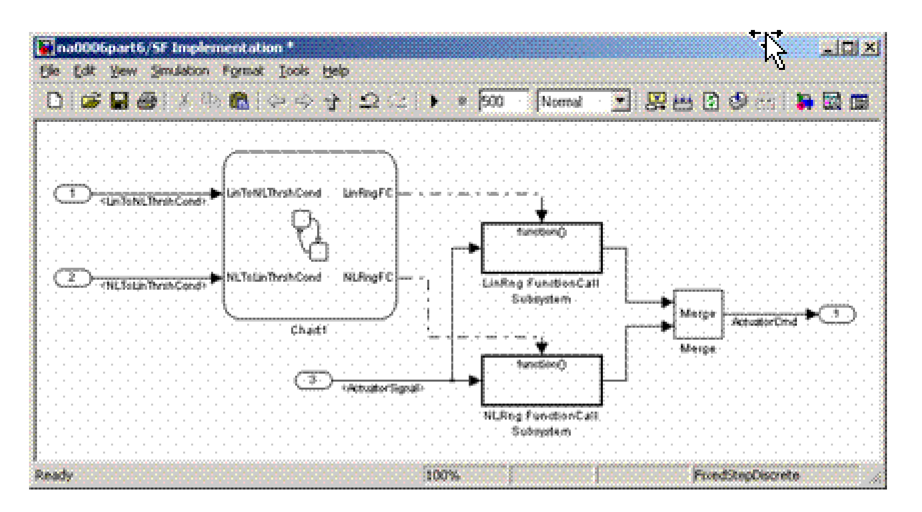
na_0006: Guidelines for mixed use of Simulink and Stateflow

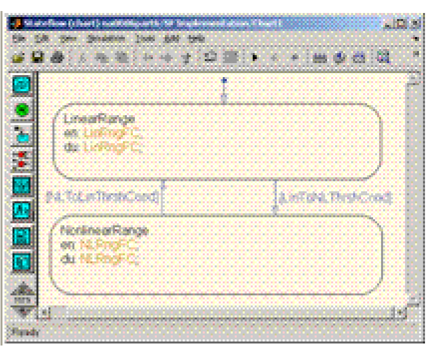




Incorrect

na_0006: Guidelines for mixed use of Simulink and Stateflow

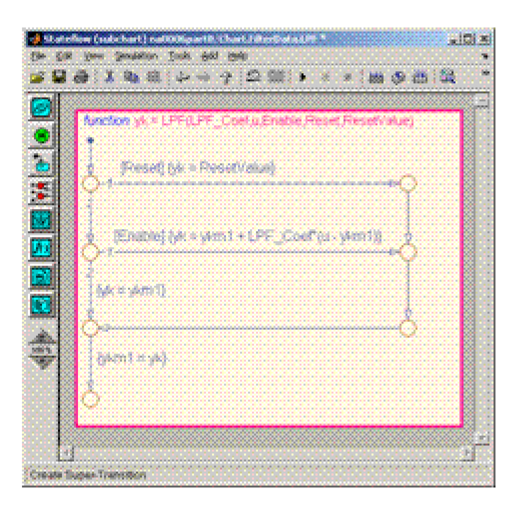




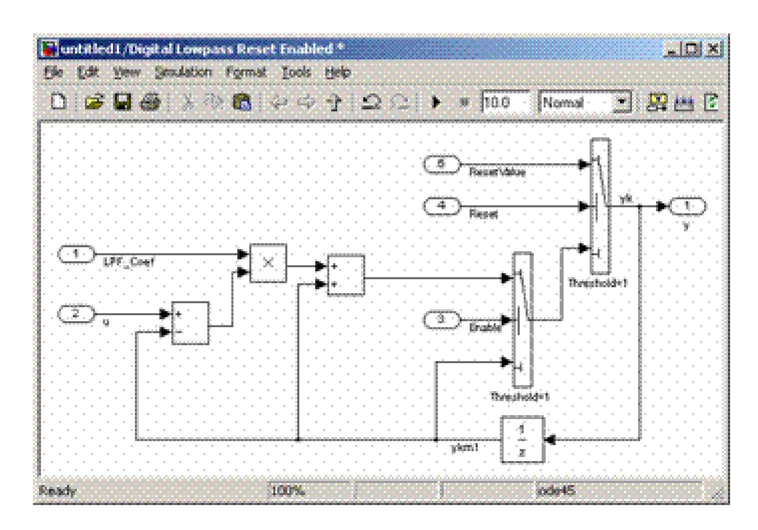
Correct

- Use Simulink to implement numerical expressions containing continuously-valued states, such as: difference equations, integrals, derivatives, and filters.

na_0006: Guidelines for mixed use of Simulink and Stateflow



Incorrect



Correct

Rationale

- Readability
- Workflow

- Simulation
- Verification and Validation
- Code Generation

Last Changed

V2.0

Model Advisor Check

Not applicable

na_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines

ID: Title

na_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines

Priority

Strongly recommended

Scope

MAAB

MATLAB Versions

All

Prerequisites

`na_0006: Guidelines for mixed use of Simulink and Stateflow`

Description

Within Stateflow, the choice of whether to use a flow chart or a state chart to model a given portion of the control algorithm functionality should be driven by the nature of the behavior being modeled.

- If the primary nature of the function segment is to calculate modes of operation or discrete-valued states, use state charts. Some examples are:
  - Diagnostic models with pass, fail, abort, and conflict states
  - Model that calculates different modes of operation for a control algorithm
- If the primary nature of the function segment involves `if-then-else` statements, use flowcharts or truth tables.

Specifics

If the primary nature of a function segment is to calculate modes or states, but `if-then-else` statements are required, add a flow chart to a state within the state chart. (See "Flowchart Patterns" on page 6-49.)

Rationale

- Readability
- Workflow

- Simulation
- Verification and Validation
- Code Generation

Last Changed

V2.0

Model Advisor Check

Not applicable

na_0007: Guidelines for use of Flow Charts, Truth Tables and State Machines

Subsystem Hierarchies

- `db_0143: Similar block types on the model levels`
- `db_0144: Use of Subsystems`
- `db_0040: Model hierarchy`

db_0143: Similar block types on the model levels

ID: Title db_0143: Similar block types on the model levels

Priority Strongly recommended

Scope NA-MAAB

MATLAB Versions All

Prerequisites None

Description You must design every level of a model with building blocks of the same type; only subsystems or only "Basic Blocks".

Blocks that You Can Place at any Model Level

Block	Example
Bus Creator	
Bus Selector	
Data Store Memory	A
Demux	
Enable (not on highest model level)	

Blocks that You Can Place at any Model Level (Continued)

Block	Example
From	
Goto	
Ground	
If	
Inport	
Merge	
Multiport Switch	
Mux	
Outport	

Blocks that You Can Place at any Model Level (Continued)

Block	Example
Rate Transition	
Selector	
Switch	
Switch Case	
Terminator	
Trigger	
Type Conversion	Convert
Unit Delay	1/z

Rationale

- Readability
- Workflow
- Verification and Validation

db_0143: Similar block types on the model levels

Last Changed	V2.1
Model Advisor Check	**By Task > Modeling Standards for MAAB > Simulink > "Check for mixing basic blocks and subsystems"**

ID: Title	db_0144: Use of Subsystems
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None
Description	Group blocks in a Simulink diagram together into subsystems based on functional decomposition of the algorithm, or portion thereof, represented in the diagram. Avoid grouping blocks into subsystems primarily for saving space in the diagram. Each subsystem in the diagram should represent a unit of functionality required to accomplish the purpose of the model or submodel.
Rationale	• Readability • Workflow • Verification and Validation • Code Generation
Last Changed	V2.0
Model Advisor Check	Not applicable

db_0040: Model hierarchy

ID: Title	db_0040: Model hierarchy
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None
Description	The model hierarchy should correspond to the functional structure of the control system.
Rationale	• Readability • Workflow • Verification and Validation • Code Generation
Last Changed	V2.0
Model Advisor Check	Not applicable

J-MAAB Model Architecture Decomposition

- `jc_0301: Controller model`
- `jc_0311: Top layer/root level`
- `jc_0321: Trigger layer`
- `jc_0331: Structure layer`
- `jc_0341: Data flow layer`

jc_0301: Controller model

ID: Title jc_0301: Controller model

Priority Mandatory

Scope J-MAAB

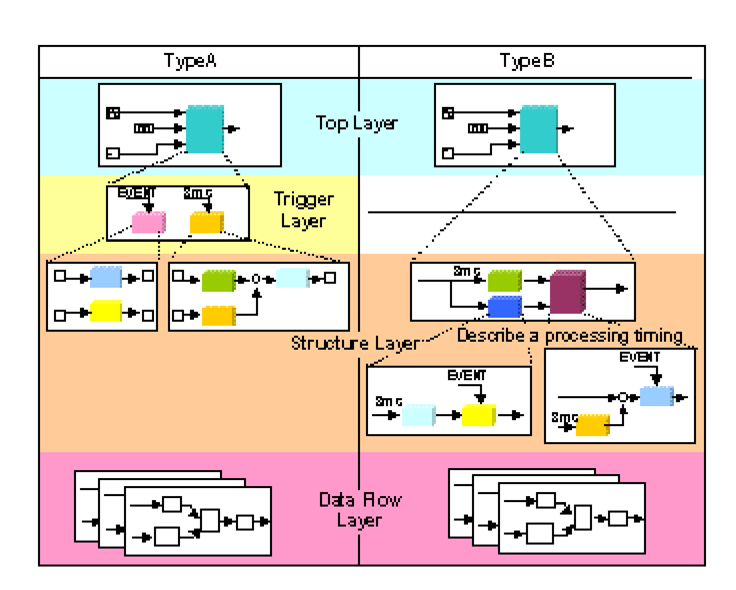
MATLAB Versions All

Prerequisites None

Description Control models are organized using the following hierarchical structure. Details on each layer are provided in corresponding rules.

- Top layer (root level), `jc_0311: Top layer/root level`
- Trigger layer, `jc_0321: Trigger layer`
- Structure layer. `jc_0331: Structure layer`
- Data flow layer, `jc_0341: Data flow layer`

Use of the Trigger level is optional. In the following figure, Type A shows the use of a trigger level while Type B shows a model without a trigger level.



Controller Model

Rationale Workflow

Last Changed V2.0

Model Advisor Check Not applicable

jc_0311: Top layer/root level

ID: Title jc_0311: Top layer/root level

Priority Mandatory

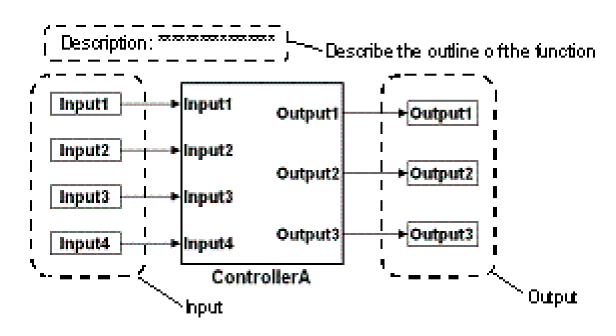
Scope J-MAAB

MATLAB Versions All

Prerequisites None

Description Items to describe in a top layer are as follows:

- Overview: Explanation of model feature overview
- Input: Input variables
- Output: Output variables



Top Layer Example

Rationale Workflow

Last Changed V2.0

Model Advisor Check

Not applicable

jc_0321: Trigger layer

ID: Title

jc_0321: Trigger layer

Priority

Mandatory

Scope

J-MAAB

MATLAB Versions

All

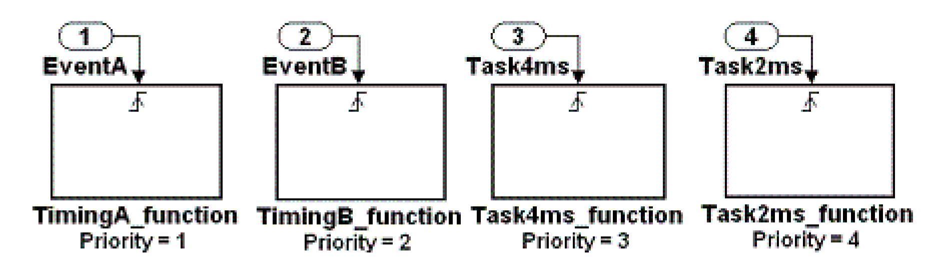
Prerequisites

None

Description

A trigger layer indicates the processing timing by using Triggered Subsystem or Function-Call Subsystem blocks.

- The blocks should set Priority, if needed.
- The priority value must be displayed as a block annotation. You should be able to understand the priority-based order without having to open the block.



Trigger Layer Example

Rationale

- Readability
- Workflow
- Code Generation

Last Changed	V2.0
Model Advisor Check	Not applicable

jc_0331: Structure layer

ID: Title jc_0331: Structure layer

Priority Mandatory

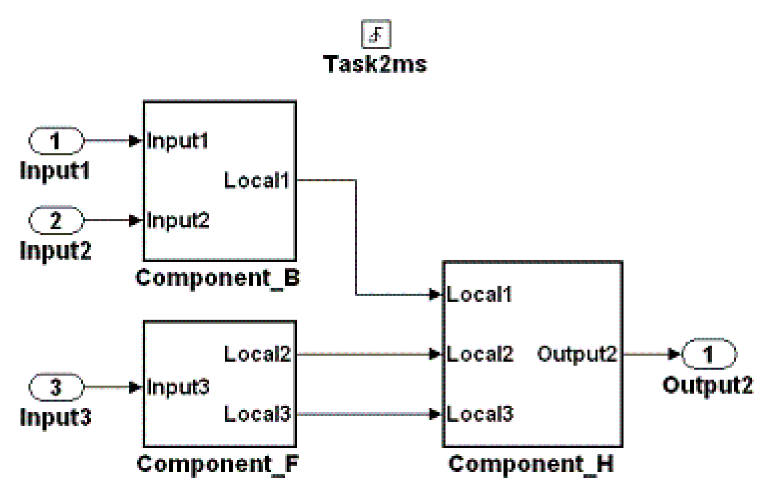
Scope J-MAAB

MATLAB Versions All

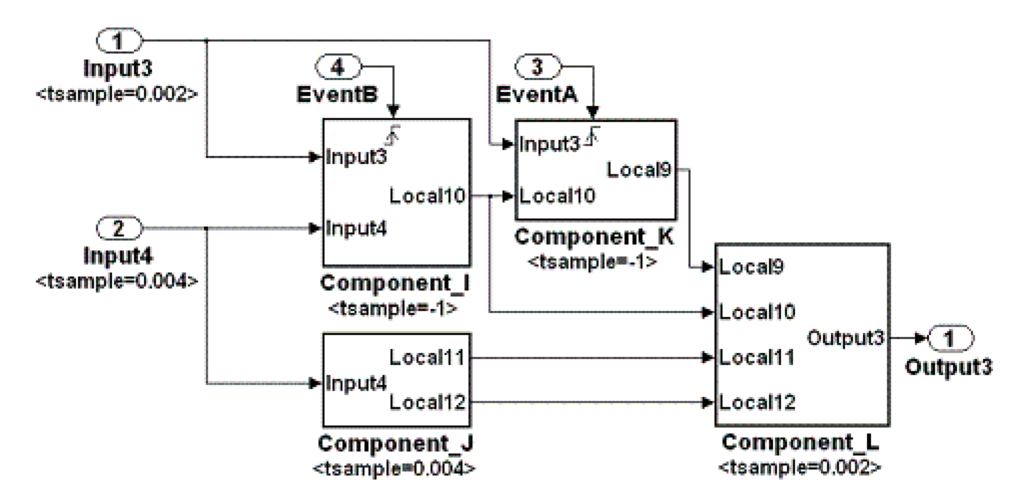
Prerequisites None

Description

- Describe a structure layer like the following structure layer example.
  - In the case of Type B, specify sample time at an Inport block or a Subsystem block to define task time of the subsystem.
  - In the case of Type B, use a block annotation at an Inport block or a Subsystem block and display sample time to clarify task time of the subsystem.
- A subsystem of a structure layer should be an atomic subsystem.



Structure Layer Example (Type A: No Description of Processing Timing)



Structure Layer Example (Type B: Description of Processing Timing)

Rationale

- Readability
- Workflow
- Code Generation

Last Changed

V2.0

Model Advisor Check

Not applicable

jc_0341: Data flow layer

ID: Title jc_0341: Data flow layer

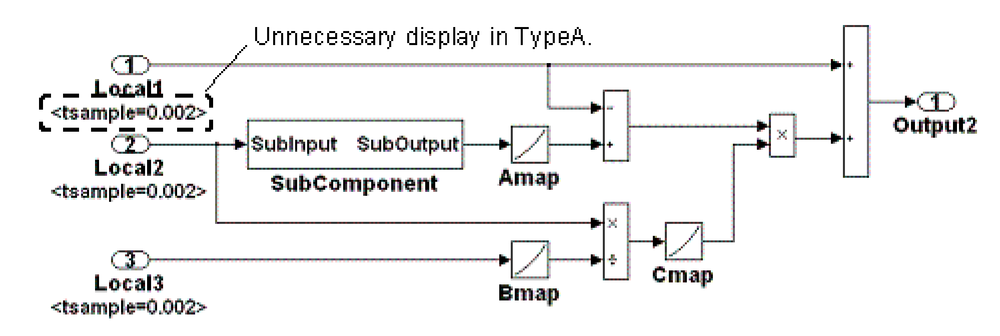
Priority Mandatory

Scope J-MAAB

MATLAB Versions All

Prerequisites None

Description Describe a data flow layer as in the following example. In the case of Type A, use a block annotation at an Inport block and display its sample time to clarify execution timing of the signal.



Data Flow Layer Example

Rationale Workflow

Last Changed V2.0

Model Advisor Check Not applicable

4

Model Configuration Options

Model Configuration Options

- `jc_0011: Optimization parameters for Boolean data types`
- `jc_0021: Model diagnostic settings`

jc_0011: Optimization parameters for Boolean data types

ID: Title jc_0011: Optimization parameters for Boolean data types

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites `na_0002: Appropriate implementation of fundamental logical and numerical operations`

Description The optimization option for Boolean data types must be enabled (on).

MATLAB Version	Option Name
R13SP2 and earlier	Boolean Logic signals
R14 and later	Use logic signals as Boolean data. (versus double)

Rationale

- Workflow
- Code Generation

Last Changed V2.0

Model Advisor Check **By Task > Modeling Standards for MAAB > Simulink > "Check Implement logic signals as Boolean data (vs. double)"**

jc_0021: Model diagnostic settings

ID: Title jc_0021: Model diagnostic settings

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description The following diagnostics must be enabled. An enabled diagnostic is set to `warning` or `error`. Setting the diagnostic option to `none` is not permitted. Diagnostics that are not listed may be set to any value (`none`, `warning`, or `error`).

Solver Diagnostics

- Algebraic loop
- Minimize algebraic loop

Sample Time Diagnostics

- Multitask rate transition

Data Validity Diagnostics

- Inf or NaN block output
- Duplicate data store names

Connectivity

- Unconnected block input ports
- Unconnected block output ports
- Unconnected line
- Unspecified bus object at root Outport block

- Mux blocks used to create bus signals
- Invalid function-call connection
- Element name mismatch

Rationale

- Workflow
- Code Generation

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Model Configuration Options > "Check model diagnostic parameters"

jc_0021: Model diagnostic settings

5

Simulink

Diagram Appearance

- na_0004: Simulink model appearance
- db_0043: Simulink font and font size
- db_0042: Port block in Simulink models
- na_0005: Port block name visibility in Simulink models
- jc_0081: Icon display for Port block
- jm_0002: Block resizing
- db_0142: Position of block names
- jc_0061: Display of block names
- db_0146: Triggered, enabled, conditional Subsystems
- db_0140: Display of basic block parameters
- jm_0013: Annotations
- db_0032: Simulink signal appearance
- db_0141: Signal flow in Simulink models
- jc_0171: Maintaining signal flow when using Goto and From blocks
- jm_0010: Port block names in Simulink models
- jc_0281: Naming of Trigger Port block and Enable Port block

na_0004: Simulink model appearance

ID: Title na_0004: Simulink model appearance

Priority Recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description The model appearance settings should conform to the following guidelines when the model is released. You can change the settings during the development process.

View Options	Setting
Model Browser	Unchecked
Screen color	White
Status Bar	Checked
Toolbar	Checked
Zoom factor	Normal (100%)

Block Display Options	Setting
Background Color	White
Foreground Color	Black
Execution Context Indicator	Unchecked
Library Link Display	None
Linearization Indicators	Checked
Model/Block I/O Mismatch	Unchecked
Model Block Version	Unchecked

Block Display Options	Setting
Sample Time Colors	Unchecked
Sorted Order	Unchecked

Signal Display Options	Setting
Port Data Types	Unchecked
Signal Dimensions	Unchecked
Storage Class	Unchecked
Test point Indicators	Checked
Viewer Indicators	Checked
Wide Nonscalar Lines	Checked

Rationale

- Readability
- Workflow

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > "Check for Simulink diagrams using nonstandard display attributes"

db_0043: Simulink font and font size

ID: Title	db_0043: Simulink font and font size
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None

Description

All text elements (block names, block annotations, and signal labels) except free text annotations within a model, must have the same font style and font size. Select font style and font size for legibility.

Note The selected font should be portable (for example, the Simulink and Stateflow default font) or convertible between platforms (for example, Arial or Helvetica 12pt).

Rationale

- Readability
- Workflow

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > "Check font formatting"

db_0042: Port block in Simulink models

ID: Title db_0042: Port block in Simulink models

Priority Strongly recommended

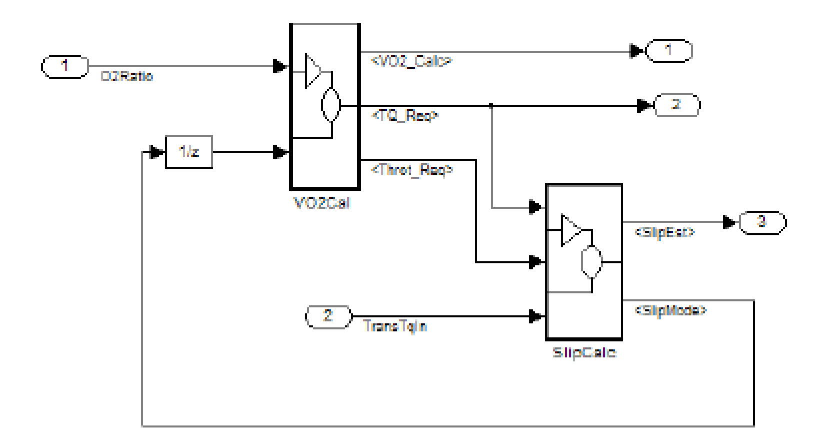
Scope MAAB

MATLAB Versions All

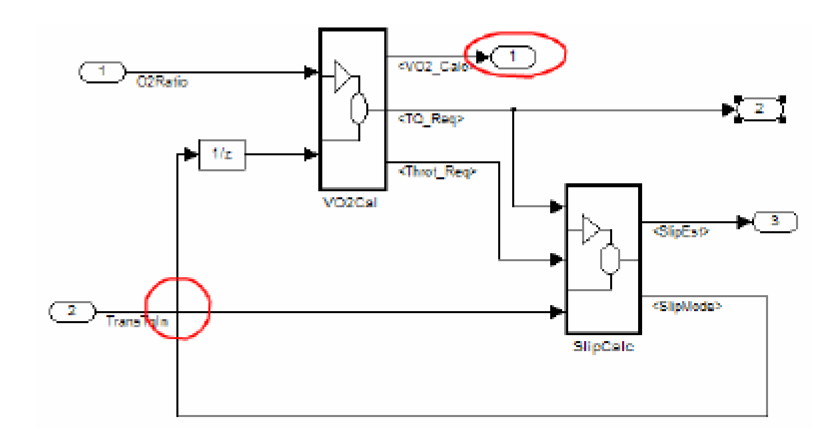
Prerequisites None

Description In a Simulink model, ports must comply with the following rules:

- Place Inport blocks on the left side of the diagram; you may move them to prevent signal crossings.
- Place Outport blocks on the right side of the diagram; you may move them to prevent signal crossings.
- You may use duplicate Inport blocks at the subsystem level, if required, but avoid doing so, if possible.
  - Do not use duplicate Inport blocks at the root level.



Correct



Incorrect

Notes on the incorrect model

- Inport 2 should be moved in so it does not cross the feedback loop lines.
- Outport 1 should be moved to the right side of the diagram.

db_0042: Port block in Simulink models

Rationale Readability

Last Changed V2.0

Model Advisor Check **By Task > Modeling Standards for MAAB > Simulink > "Check positioning and configuration of ports"**

na_0005: Port block name visibility in Simulink models

ID: Title

na_0005: Port block name visibility in Simulink models

Priority

Strongly recommended

Scope

MAAB

MATLAB Versions

All

Prerequisites

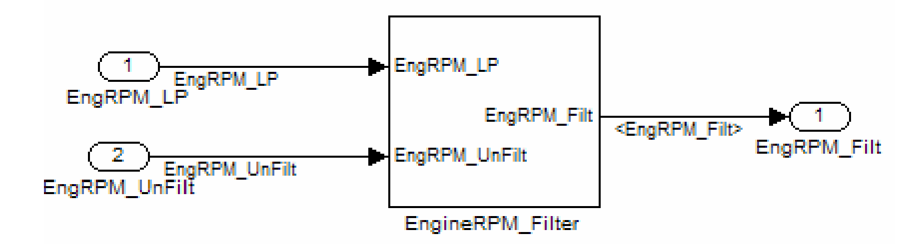
None

Description

While for some items, it is not possible to define a single approach that may apply to all organizations' internal processes, it is important that, at least within a given organization, a single consistent approach is followed. An organization applying the guidelines must enforce one of the following alternatives.

Apply one of the following practices:

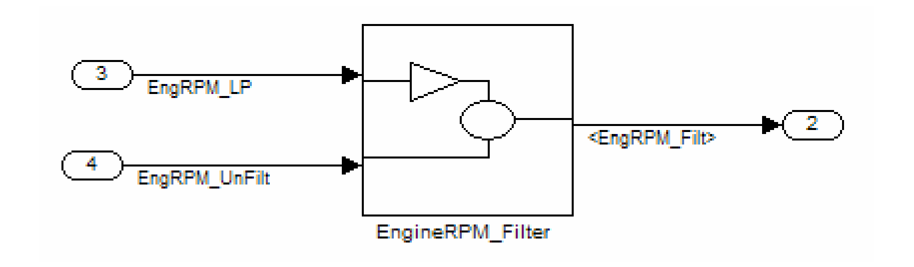
- The name of an Inport or Outport block is not hidden. (**Format > Hide Name** is not allowed.)



- The name of an Inport or Outport block must be hidden. (**Format > Hide Name** is used.)

  Exception: The names cannot be hidden inside library subsystem blocks.

na_0005: Port block name visibility in Simulink models



Rationale Readability

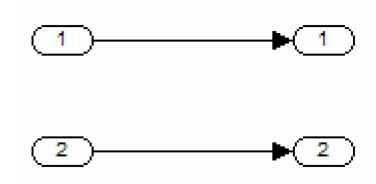
Last Changed V2.0

Model Advisor Check **By Task > Modeling Standards for MAAB > Simulink > "Check visibility of block port names"**

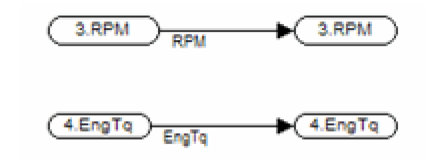
ID: Title	jc_0081: Icon display for Port block
Priority	Recommended
Scope	MAAB
MATLAB Versions	R14 and later
Prerequisites	None

Description

The Icon display setting should be set to Port number for Inport and Outport blocks.



Correct



Incorrect

Rationale	Readability
Last Changed	V2.0

jc_0081: Icon display for Port block

Model Advisor Check

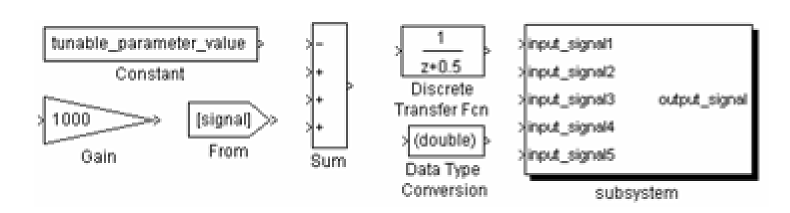
By Task > Modeling Standards for MAAB > Simulink > "Check for unconnected ports and signal lines"

ID: Title	jm_0002: Block resizing
Priority	Mandatory
Scope	MAAB
MATLAB Versions	All
Prerequisites	None

Description

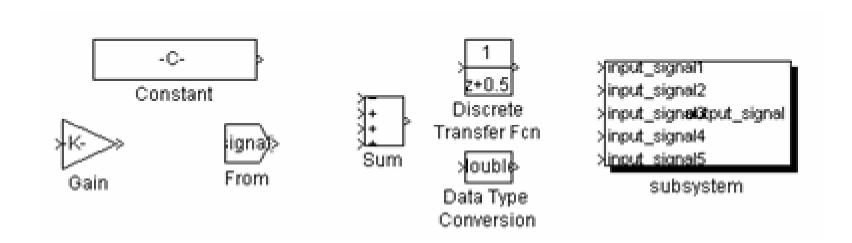
All blocks in a model must be sized such that the icon is completely visible and recognizable. In particular, any displayed text (for example, tunable parameters, file names, or equations) in the icon must be readable.

This guideline requires that you resize blocks with variable icons or blocks with a variable number of inputs and outputs. In some cases, it may not be practical or desirable to resize the icon of a subsystem block so that all of the input and output names within it are readable. In such cases, you may hide the names in the icon by using a mask or by hiding the names in the subsystem associated with the icon. If you do this, the signal lines coming into and out of the subsystem block should be clearly labeled in close proximity to the block.



Correct

jm_0002: Block resizing



Incorrect

Rationale Readability

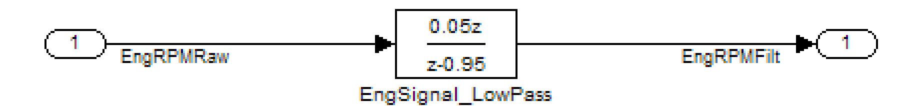
Last Changed V2.0

Model Advisor Check Not applicable

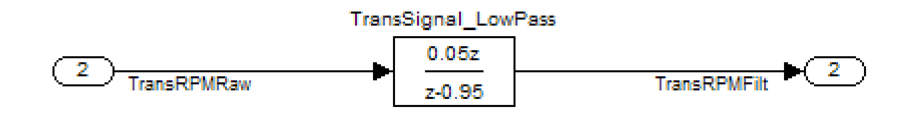
ID: Title	db_0142: Position of block names
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None

Description

If shown, place the name of a block below the block.



Correct



Incorrect

Rationale

- Readability
- Workflow

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > "Check whether block names appear below blocks"

jc_0061: Display of block names

ID: Title jc_0061: Display of block names

Priority Recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description

- Display a block name when it provides descriptive information.
- Do not display a block name if the block function is known and understood from the block appearance.

Rationale Readability

Last Changed V2.0

Model Advisor Check **By Task > Modeling Standards for MAAB > Simulink > "Check the display attributes of block names"**

db_0146: Triggered, enabled, conditional Subsystems

ID: Title db_0146: Triggered, enabled, conditional Subsystems

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description Place blocks that are inside of and define subsystems as conditional or iterative at a consistent location at the top of the subsystem diagram. This applies to the following types of blocks:

- Function call
- Enabled
- Triggered
- If /Else Action

Rationale

- Readability
- Workflow
- Verification and Validation

Last Changed V2.0

Model Advisor Check **By Task > Modeling Standards for MAAB > Simulink > "Check for incorrect position of Trigger and Enable blocks"**

db_0140: Display of basic block parameters

ID: Title db_0140: Display of basic block parameters

Priority Recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description If the block parameter meets all of the following conditions, display it as a text annotation to the block:

- Modified from the default setting.
- Affects either the simulation or code generation behavior of a block.
- Not automatically displayed as part of the block icon.
- Rarely changes.

Examples

Block: Parameter	Annotate Block	Rationale
Gain: Gain	N	The Gain parameter is visible in the block icon.
Integrator: Show State Port	N	The Show State Port parameter is visible as a modification to the block.
Integrator: Initial condition	Y	The Initial condition parameter affects the simulation and/or code generation behavior. It is not visible in the block icon.

Examples (Continued)

Block: Parameter	Annotate Block	Rationale
Sample Time	Y/N	If the sample time that you select matches the subsystem sample time, you do not need to display the rate information.
Saturation: Upper Limit	N	Annotating the block with the Saturation: Upper Limit value is optional because the value will change.

Note You can use the attribute string to support the display of basic block parameters. Add the attribute information that you want using the block annotation tab.

Rationale

- Readability
- Verification and Validation

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > "Check for nondefault block attributes"

jm_0013: Annotations

ID: Title jm_0013: Annotations

Priority Strongly recommended

Scope MAAB

MATLAB Versions R12.1

Prerequisites None

Description Annotations should not have a drop shadow. (**Format > Show Drop Shadow** is not allowed.)

This is a correct annotation

This is an incorrect annotation

Rationale Readability

Last Changed V2.0

Model Advisor Check **By Task > Modeling Standards for MAAB > Simulink > "Check for annotations with drop shadows"**

ID: Title

db_0032: Simulink signal appearance

Priority

Strongly recommended

Scope

MAAB

MATLAB Versions

All

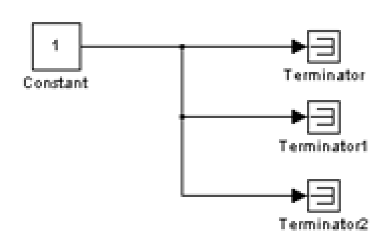
Prerequisites

None

Description

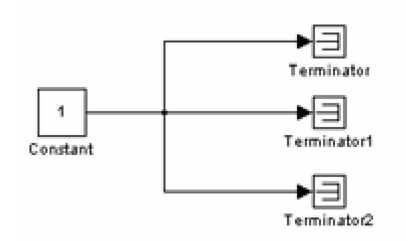
Signal lines

- Should not cross each other, if possible
- Are drawn with right angles
- Are not drawn one upon the other
- Do not cross any blocks
- Should not split into more than two sublines at a single branching point



Correct

db_0032: Simulink signal appearance



Incorrect

Rationale

- Readability
- Workflow

Last Changed

V2.0

Model Advisor Check

Not applicable

db_0141: Signal flow in Simulink models

ID: Title db_0141: Signal flow in Simulink models

Priority Strongly recommended

Scope MAAB

Versions All

Prerequisites None

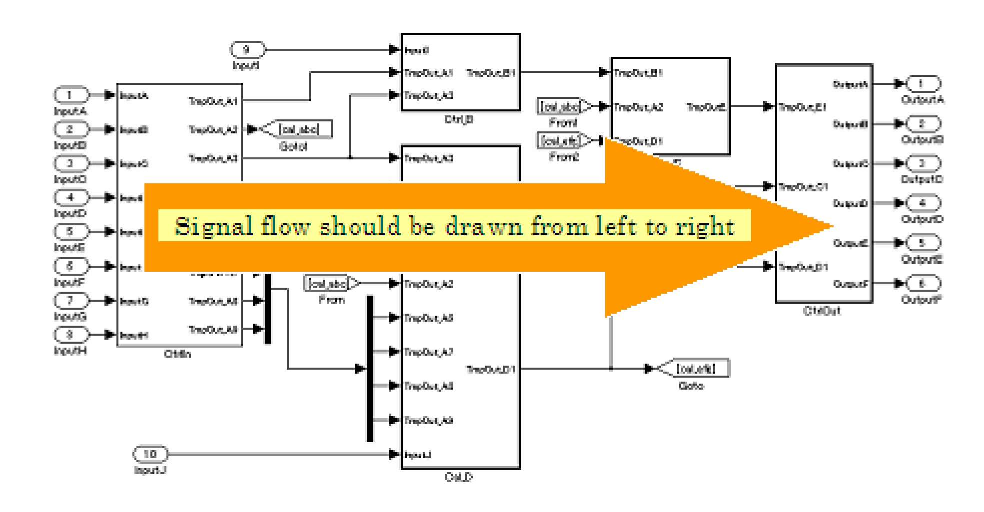
Description The signal flow in a model is from left to right.

Exception: Feedback loops

Sequential blocks or subsystems are arranged from left to right.

Exception: Feedback loops

Parallel blocks or subsystems are arranged from top to bottom.



Rationale

- Readability
- Workflow

- Verification and Validation

Last Changed

V2.0

Model Advisor Check

Not applicable

jc_0171: Maintaining signal flow when using Goto and From blocks

ID: Title jc_0171: Maintaining signal flow when using Goto and From blocks

Priority Strongly recommended

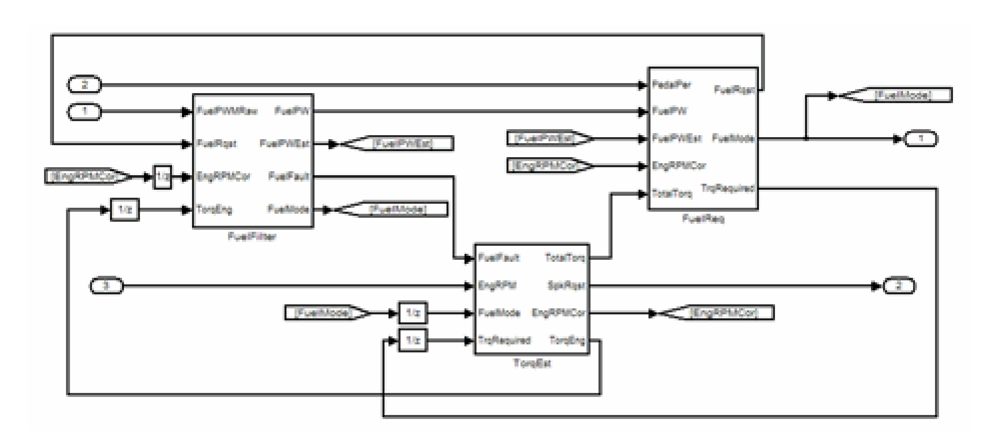
Scope MAAB

MATLAB Versions All

Prerequisites None

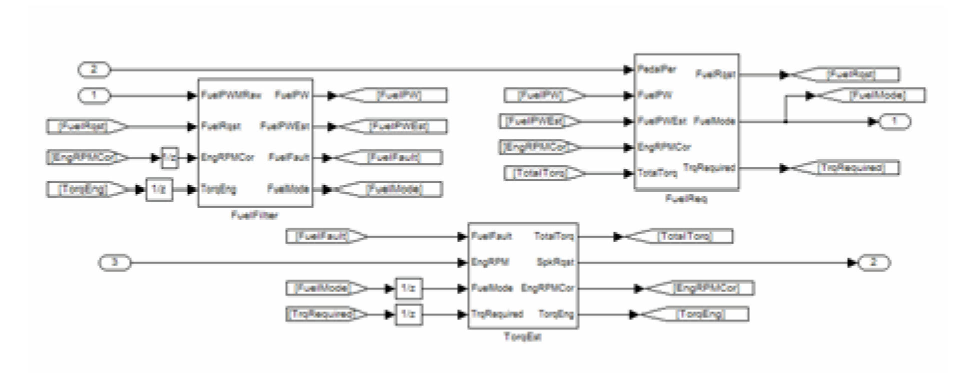
Description

- You must maintain visual depiction of signal flow between subsystems.
- You can use Goto and From blocks provided that you use at least one signal line between connected subsystems.
- If the subsystems are connected in a feed-forward and feedback loop, you must connect at least one signal line for each direction.



Correct

jc_0171: Maintaining signal flow when using Goto and From blocks



Incorrect

Rationale

- Readability
- Workflow
- Verification and Validation

Last Changed

V2.0

Model Advisor Check

Not applicable

jm_0010: Port block names in Simulink models

ID: Title

jm_0010: Port block names in Simulink models

Priority

Strongly recommended

Scope

MAAB

MATLAB Versions

All

Prerequisites

- `db_0042:  Port block in Simulink models`
- `na_0005:  Port block name visibility in Simulink models`

Description

For some items, though you may not be able to define a single approach for internal processes of all organizations, within a given organization, try to follow a single, consistent approach. An organization applying the guidelines must enforce *one* of the following options:

- **Names of Inport and Outport blocks must match corresponding signal or bus names.**

  Exceptions:

  - When any combination of an Inport block, an Outport block, and any other block have the same block name, use a suffix or prefix on the Inport and Outport blocks.
  - One common suffix / prefix is `_in` for Inport blocks and `_out` for Outport blocks.
  - You may use any suffix or prefix on the ports, however, the prefix that you select must be consistent.
  - Library blocks and reusable subsystems that encapsulate generic functionality.

- **When names of Inport and Outport blocks are hidden, apply a consistent naming practice for the blocks.** Suggested practices include leaving the default names (for example, `Out1`), giving them

the same name as the associated signal, or giving them a shortened or mangled version of the name of the associated signal.

Rationale

- Readability
- Workflow
- Simulation

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > "Check for matching port and signal names"

jc_0281: Naming of Trigger Port block and Enable Port block

ID: Title jc_0281: Naming of Trigger Port block and Enable Port block

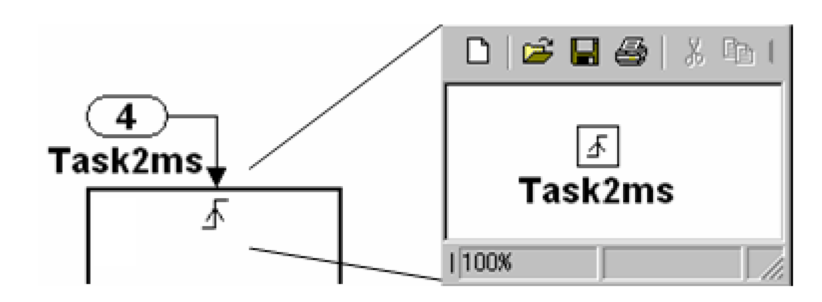
Priority Strongly recommended

Scope J-MAAB

MATLAB Versions All

Prerequisites None

Description For Trigger and Enable port blocks, match the block name of the signal triggering the subsystem.



Rationale Readability

Last Changed V2.0

Model Advisor Check **By Task > Modeling Standards for MAAB > Simulink > "Check Trigger and Enable block names"**

Signals

- `na_0008: Display of labels on signals`
- `na_0009: Entry versus propagation of signal labels`
- `db_0097: Position of labels for signals and busses`
- `db_0081: Unconnected signals, block inputs and block outputs`

The preceding guidelines apply to signals and signal labels. For background information, see “Signals and Signal Labels” on page D-3.

Some of the preceding guidelines refer to basic blocks. For an explanation of the meaning and some examples, see “Basic Blocks” on page D-2.

na_0008: Display of labels on signals

ID: Title	na_0008: Display of labels on signals
Priority	Recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None

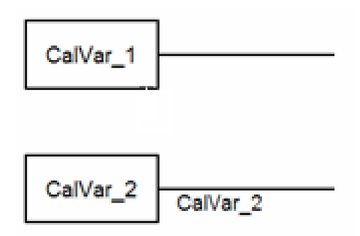
Description

- A label must be displayed on a signal originating from the following blocks:
  - Inport block
  - From block (block icon exception applies – see the Note below
  - Data Store Read block (block icon exception applies)
  - Subsystem block or Stateflow chart block (block icon exception applies)
  - Constant block (block icon exception applies)
  - Bus Selector block (the tool forces this to happen)
  - Demux block
  - Selector block
- A label must be displayed on any signal connected to the following destination blocks (directly or by way of a basic block that performs a nontransformative operation):
  - Outport block
  - Goto block
  - Data Store Write block
  - Bus Creator block
  - Mux block
  - Subsystem block

  - Chart block

Note Block icon exception (applicable only where called out): If the signal label is visible in the originating block icon display, the connected signal does not need to have the label displayed, unless the signal label is needed elsewhere due to a destination-based rule.

- In addition, a label may be displayed on any other signal of interest to you or your customers.



Rationale

- Readability
- Workflow
- Verification and Validation
- Code Generation

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > "Check for proper labeling on signal lines"

na_0009: Entry versus propagation of signal labels

ID: Title na_0009: Entry versus propagation of signal labels

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites `na_0008: Display of labels on signals`

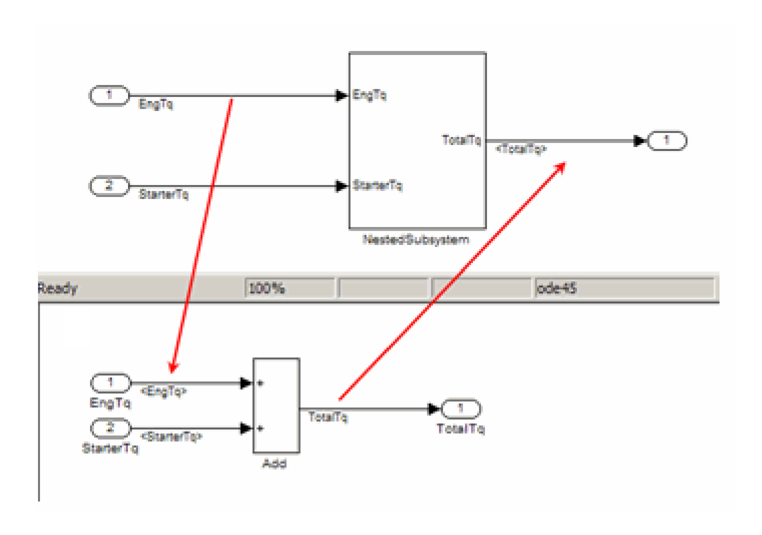
Description If a label is present on a signal, the following rules define whether that label is created there (entered directly on the signal) or propagated from its true source (inherited from elsewhere in the model by using the less than (<) character).

- Any displayed signal label must be *entered* for signals that:
  - Originate from an Inport at the Root (top) Level of a model
  - Originate from a basic block that performs a transformative operation (For the purpose of interpreting this rule only, the Bus Creator block, Mux block, and Selector block are considered to be included among the blocks that perform transformative operations.)
- Any displayed signal label must be *propagated* for signals that:
  - Originate from an Inport block in a nested subsystem

    Exception: If the nested subsystem is a library subsystem, a label may be entered on the signal coming from the Inport to accommodate reuse of the library block.
  - Originate from a basic block that performs a nontransformative operation
  - Originate from a Subsystem or Stateflow chart block

na_0009: Entry versus propagation of signal labels

Exception: If the connection originates from the output of a library subsystem block instance, a new label may be entered on the signal to accommodate reuse of the library block.



Rationale

- Readability
- Workflow
- Verification and Validation
- Code Generation

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > "Check for propagated signal labels"

db_0097: Position of labels for signals and busses

ID: Title

db_0097: Position of labels for signals and busses

Priority

Strongly recommended

Scope

MAAB

MATLAB Versions

All

Prerequisites

None

Description

The labels must be visually associated with the corresponding signal and not overlap other labels, signals, or blocks.

Labels should be located consistently below horizontal lines and close to the corresponding source or destination block.

Rationale

- Readability
- Workflow

Last Changed

V2.0

Model Advisor Check

Not applicable

db_0081: Unconnected signals, block inputs and block outputs

ID: Title
db_0081: Unconnected signals, block inputs and block outputs

Priority
Mandatory

Scope
MAAB

MATLAB Versions
All

Prerequisites
None

Description
A system must not have any:

- Unconnected subsystem or basic block inputs
- Unconnected subsystem or basic block outputs
- Unconnected signal lines

In addition:

- An otherwise unconnected input should be connected to a ground block
- An otherwise unconnected output should be connected to a terminator block

Rationale

- Readability
- Workflow
- Verification and Validation

Last Changed
V2.0

db_0081: Unconnected signals, block inputs and block outputs

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > "Check for unconnected ports and signal lines"

Block Usage

- na_0003: Simple logical expressions in If Condition block
- na_0002: Appropriate implementation of fundamental logical and numerical operations
- jm_0001: Prohibited Simulink standard blocks inside controllers
- hd_0001: Prohibited Simulink sinks
- na_0011: Scope of Goto and From blocks
- jc_0141: Use of the Switch block
- jc_0121: Use of the Sum block
- jc_0131: Use of Relational Operator block
- jc_0161: Use of Data Store Read/Write/Memory blocks

Some of the preceding guidelines refer to basic blocks. For an explanation of the meaning and some examples, see "Basic Blocks" on page D-2.

na_0003: Simple logical expressions in If Condition block

ID: Title

na_0003: Simple logical expressions in If Condition block

Priority

Mandatory

Scope

MAAB

MATLAB Versions

All

Prerequisites

None

Description

A logical expression may be implemented within an If Condition block instead of building it up with logical operation blocks, if the expression contains two or fewer primary expressions. A primary expression is defined as one of the following:

- An input
- A constant
- A constant parameter
- A parenthesized expression containing no operators except zero or one instance of the following operators: < , <= , >, >=, ~=, ==, ~. (See the following examples.)

Exception

A logical expression may contain more than two primary expressions if both of the following are true:

- The primary expressions are all inputs
- Only one type of logical operator is present

Examples of Acceptable Exceptions

- `u1 | u2 | u3 | u4 | u5`
- `u1 & u2 & u3 & u4`

na_0003: Simple logical expressions in If Condition block

Examples of Primary Expressions

- `u1`
- 5
- `K`
- (`u1` > 0)
- (`u1` <= `G`)
- (`u1` > `U2`)
- (~`u1`)

Examples of Acceptable Logical Expressions

- `u1 | u2`
- (`u1` > 0) & (`u1` < 20)
- (`u1` > 0) & (`u2` < `u3`)
- (`u1` > 0) & (~`u2`)

Examples of Unacceptable Logical Expressions

`u1 & u2 | u3`	(too many primary expressions)
`u1 & (u2 | u3)`	(unacceptable operator within primary expression)
(`u1` > 0) & (`u1` < 20) & (`u2` > 5)	(too many primary expressions that are not inputs)
(`u1` > 0) & ((2*`u2`) > 6)	(unacceptable operator within primary expression)

Rationale

- Readability
- Workflow

na_0003: Simple logical expressions in If Condition block

Last Changed

V2.0

Model Advisor Check

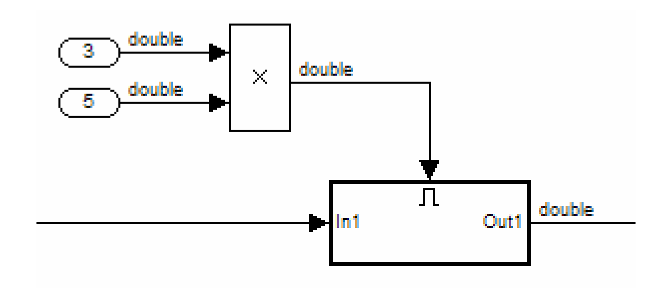
Not applicable

na_0002: Appropriate implementation of fundamental logical and numerical operations

ID: Title	na_0002: Appropriate implementation of fundamental logical and numerical operations
Priority	Mandatory
Scope	MAAB
MATLAB Versions	All
Prerequisites	None

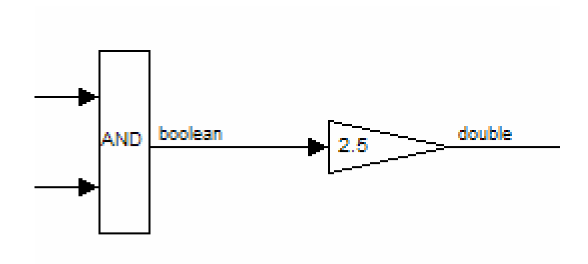
Description

- Blocks that are intended to perform numerical operations must not be used to perform logical operations.



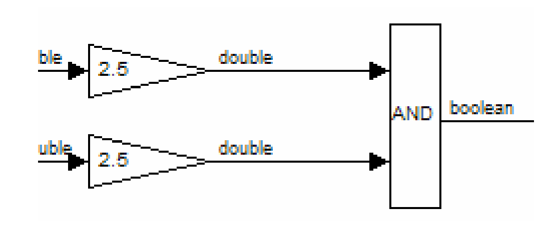
Incorrect

- A logical output should never be connected directly to the input of blocks that operate on numerical inputs.
- The result of a logical expression fragment should never be operated on by a numerical operator.



Incorrect

- Blocks that are intended to perform logical operations must not be used to perform numerical operations.
- A numerical output should never be connected to the input of blocks that operate on logical inputs.



Incorrect

Rationale

- Readability
- Workflow

Last Changed

V2.0

Model Advisor Check

Not applicable

jm_0001: Prohibited Simulink standard blocks inside controllers

ID: Title jm_0001: Prohibited Simulink standard blocks inside controllers

Priority Mandatory

Scope MAAB

MATLAB Versions All

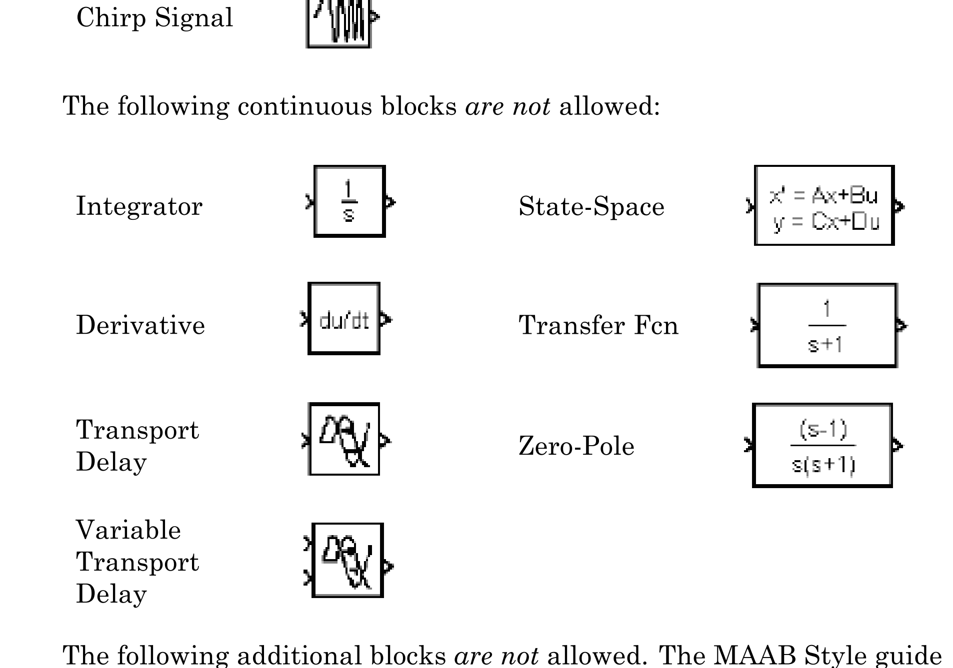
Prerequisites None

Description Controller models must be designed from discrete blocks.

The following sources *are not* allowed:

Signal Generator		Clock	
Step		Digital Clock	12:34
Ramp		From File	untitled.mat
Sine Wave		From Workspace	[T,U]
Repeating Sequence		Random Number	
Discrete Pulse Generator		Uniform Random Number	

jm_0001: Prohibited Simulink standard blocks inside controllers



The following continuous blocks *are not* allowed:

The following additional blocks *are not* allowed. The MAAB Style guide group recommends not using the following blocks. The list may be extended by individual companies.

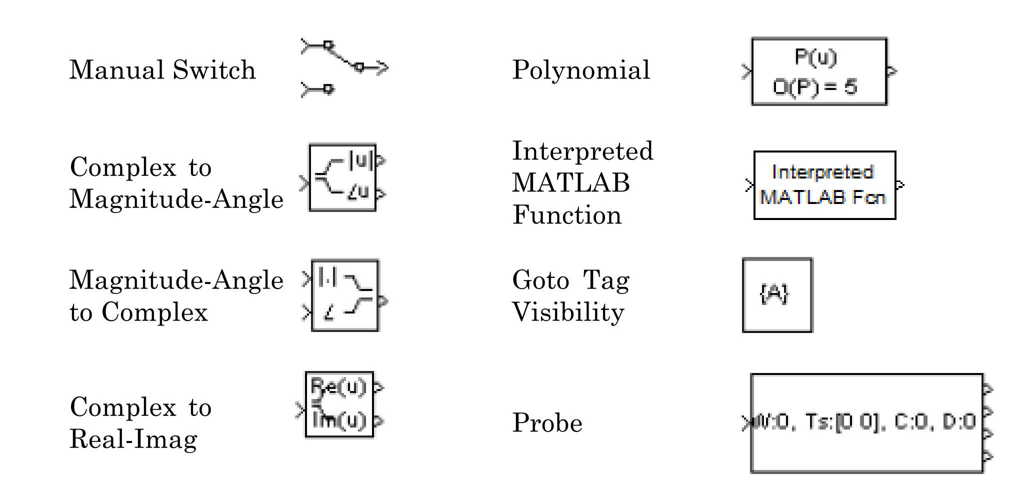
Slider Gain — 1

Real-Imag to Complex — Re, Im

Algebraic Constraint — f(z) Solve f(z) = 0 z

Hit Crossing

jm_0001: Prohibited Simulink standard blocks inside controllers



Rationale

- Readability
- Workflow
- Code Generation

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > "Check for prohibited blocks in discrete controllers"

hd_0001: Prohibited Simulink sinks

ID: Title hd_0001: Prohibited Simulink sinks

Priority Strongly recommended

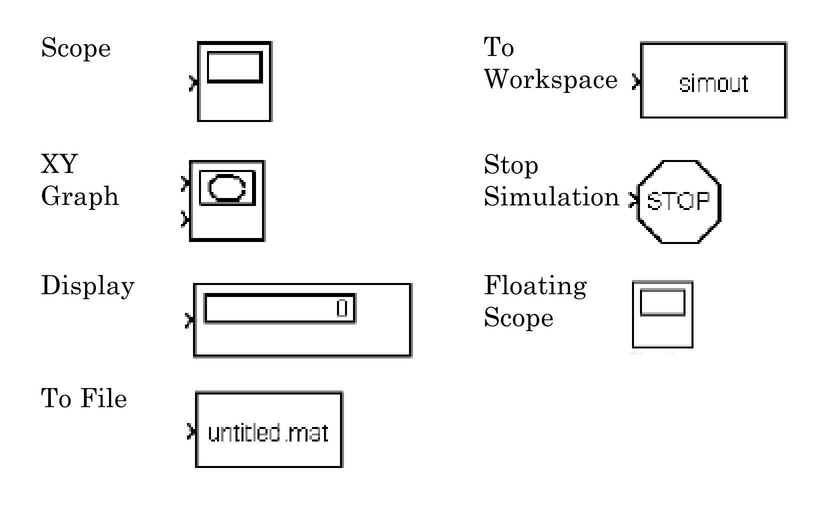
Scope MAAB

MATLAB Versions All

Prerequisites None

Description Controller models must be designed from discrete blocks.

The following sink blocks *are not* allowed:



Rationale

- Readability
- Workflow

Last Changed V2.0

hd_0001: Prohibited Simulink sinks

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > "Check for prohibited sink blocks"

na_0011: Scope of Goto and From blocks

ID: Title na_0011: Scope of Goto and From blocks

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description For signal flows, From and Goto blocks must use local scope.

Note Control flow signals may use global scope.

Rationale

- Readability
- Workflow
- Code Generation

Last Changed V2.0

Model Advisor Check **By Task > Modeling Standards for MAAB > Simulink > "Check scope of From and Goto blocks"**

jc_0141: Use of the Switch block

ID: Title jc_0141: Use of the Switch block

Priority Strongly recommended

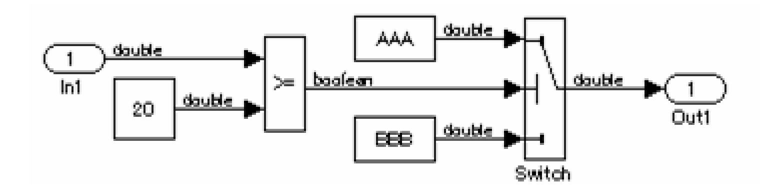
Scope MAAB

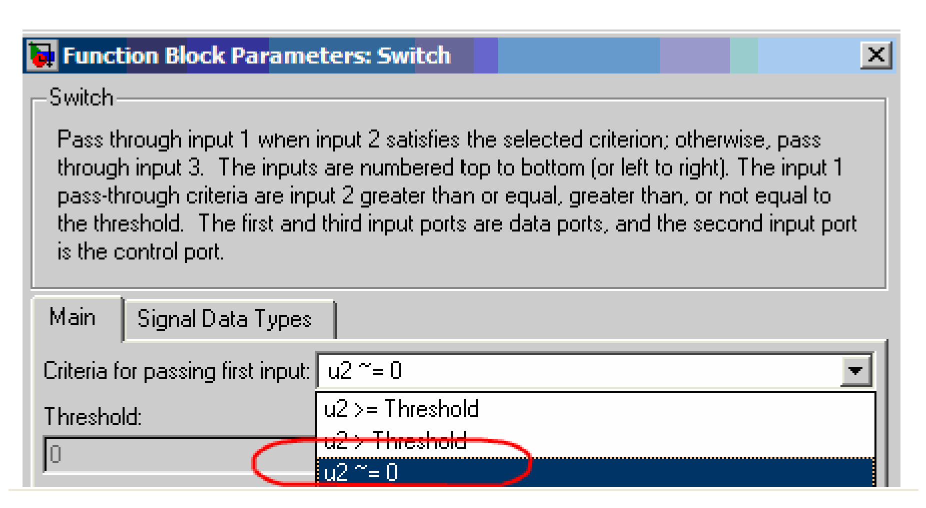
MATLAB Versions All

Prerequisites None

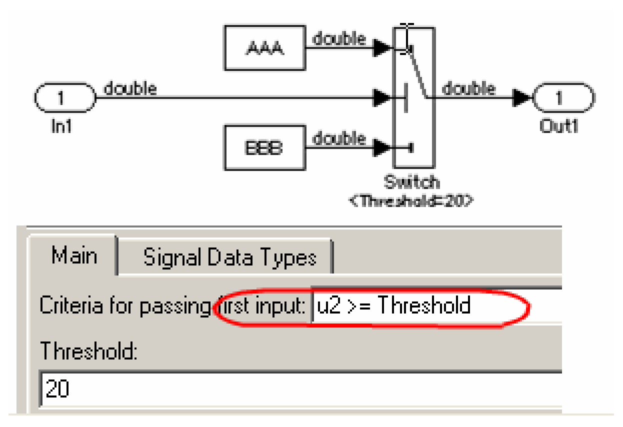
Description

- The switch condition, input 2, must be a Boolean value.
- The block parameter, **Criteria for passing first input**, should be set to `u2~=0`.
- The block parameter, **Criteria for passing first input**, must not be set to `u2>`*`threshold`* for R13 versions of MATLAB.





Correct



Incorrect

jc_0141: Use of the Switch block

Rationale

- Readability
- Workflow

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > "Check use of Switch blocks"

ID: Title jc_0121: Use of the Sum block

Priority Recommended

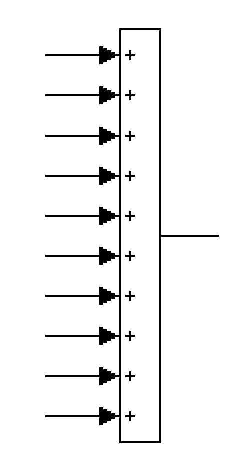
Scope MAAB

MATLAB Versions All

Prerequisites None

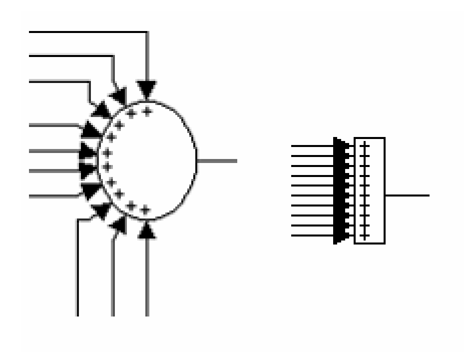
Description Sum blocks should:

- Use the “rectangular” shape.
- Be sized so that the input signals do not overlap.



Correct

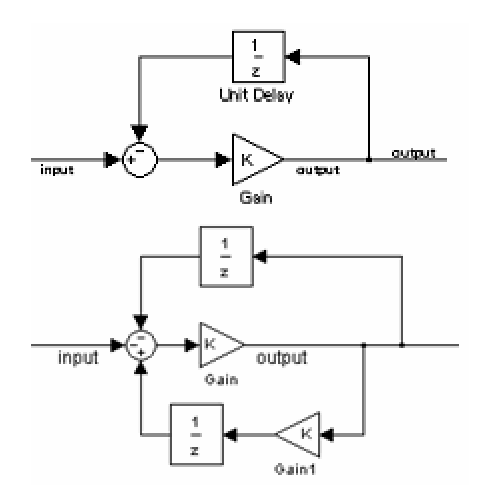
jc_0121: Use of the Sum block



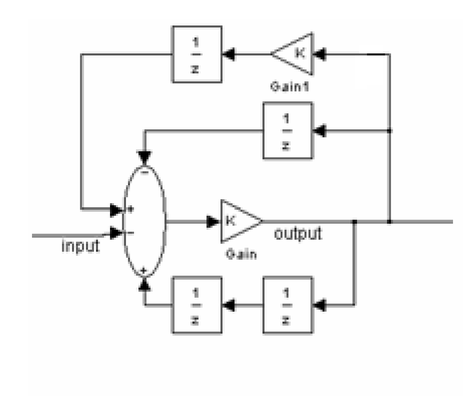
Incorrect

You may use the round shape in feedback loops.

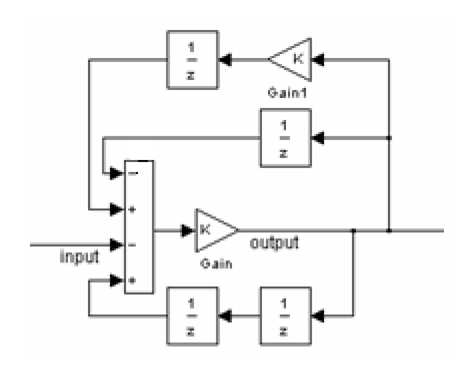
- There should be no more than three inputs.
- Position the inputs at 90,180,270 degrees.
- Position the output at 0 degrees.



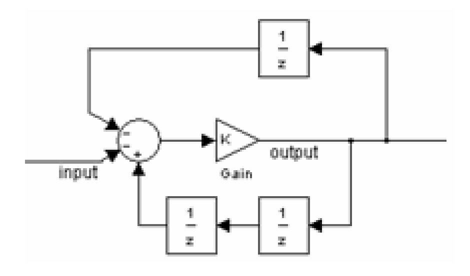
Correct



Incorrect



Correct



Incorrect

jc_0121: Use of the Sum block

Rationale Readability

Last Changed V2.0

Model Advisor Check Not applicable

ID: Title jc_0131: Use of Relational Operator block

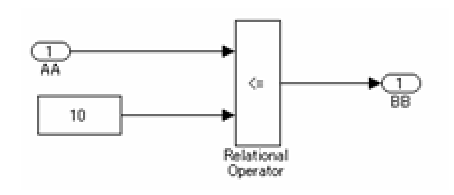
Priority Recommended

Scope J-MAAB

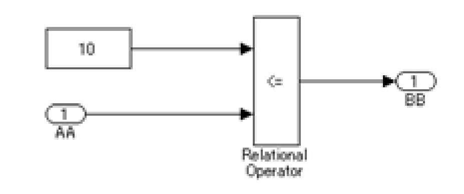
MATLAB Versions All

Prerequisites None

Description When the relational operator is used to compare a signal to a constant value, the constant input should be the second (lower) input signal.



Correct



Incorrect

Rationale

- Readability
- Code Generation

jc_0131: Use of Relational Operator block

Last Changed V2.0

Model Advisor Check **By Task > Modeling Standards for MAAB > Simulink > "Check configuration of Relational Operator blocks"**

jc_0161: Use of Data Store Read/Write/Memory blocks

ID: Title jc_0161: Use of Data Store Read/Write/Memory blocks

Priority Strongly recommended

Scope J-MAAB

MATLAB Versions All

Prerequisites `jc_0341: Data flow layer`

Description Data Store Memory, Data Store Read, and Data Store Write blocks are

- Prohibited in a data flow layer
- Allowed between subsystems running at different rates

Rationale

- Readability
- Workflow

Last Changed V2.0

Model Advisor Check Not applicable

Block Parameters

- `db_0112: Indexing`
- `na_0010: Grouping data flows into signals`
- `db_0110: Tunable parameters in basic blocks`

Some of the preceding guidelines refer to basic blocks. For an explanation of the meaning and some examples, see "Basic Blocks" on page D-2.

db_0112: Indexing

ID: Title db_0112: Indexing

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description One-based indexing [1, 2, 3,…] is for:

- MATLAB

    Workspace variables and structures
    Local variables of functions written in MATLAB-code
    Global variables

- Simulink

    Signal vectors and matrices
    Parameter vectors and matrices
    S-function input and output signal vectors and matrices in MATLAB-code
    S-function parameter vectors and matrices in MATLAB-code
    S-function local variables in MATLAB-code

Zero-based indexing [0, 1, 2, ...] is for:

- Simulink

    S-function input and output signal vectors and matrices in C
    S-function input parameters in C
    S-function parameter vectors and matrices in C
    S-function local variables in C

- Stateflow

    Input and output signal vectors and matrices

Parameter vectors and matrices
Local variables
Custom variables and structures in C

- C code

  Local variables and structures
  Global variables

Rationale

- Readability
- Workflow
- Code Generation

Last Changed

V2.1

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > "Check for indexing in blocks"

na_0010: Grouping data flows into signals

ID: Title

na_0010: Grouping data flows into signals

Priority

Strongly recommended

Scope

MAAB

MATLAB Versions

All

Prerequisites

None

Description

Vectors

The individual scalar signals composing a vector must have common functionality, data types, dimensions, and units. The most common example of a vector signal is sensor or actuator data that is grouped into an array indexed by location. The output of a Mux block must always be a vector. The inputs to a Mux block must always be scalars.

Busses

Signals that do not meet criteria for use as a vector, as previously described, must only be grouped into bus signals. Use Bus Selector blocks only with a bus signal input; do not use them to extract scalar signals from vector signals.

Examples

Some examples of vector signals include:

Vector type	Size
Row vector	[1 n]
Column vector	[n 1]
Wheel speed vector	[1 Number of wheels]
Cylinder vector	[1 Number of cylinders]

Vector type	Size
Position vector based on 2D coordinates	[1 2]
Position vector based on 3D coordinates	[1 3]

Some examples of bus signals include:

Bus type	Elements
Sensor Bus	Force Vector [Fx, Fy, Fz]
	Position
	Wheel Speed Vector $[\Theta_{lf}, \Theta_{rf}, \Theta_{lr}, \Theta_{rr}]$
	Acceleration
	Pressure
Controller Bus	Sensor Bus
	Actuator Bus
Serial Data Bus	Coolant Temperature
	Engine Speed, Passenger Door Open

Rationale

- Readability
- Workflow

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > "Check for signal bus and Mux block usage"

db_0110: Tunable parameters in basic blocks

ID: Title

db_0110: Tunable parameters in basic blocks

Priority

Strongly recommended

Scope

MAAB

MATLAB Versions

All

Prerequisites

None

Description

To ensure that a parameter is tunable, enter it in the basic block:

- Without any expression.
- Without a data type conversion.
- Without selection of rows or columns.



Correct



Incorrect

Rationale

- Readability
- Workflow
- Code Generation

Last Changed

V2.0

db_0110: Tunable parameters in basic blocks

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > "Check use of tunable parameters in blocks"

Simulink Patterns

- `na_0012: Use of Switch vs. If-Then-Else Action Subsystem`
- `db_0114: Simulink patterns for If-then-else-if constructs`
- `db_0115: Simulink patterns for case constructs`
- `db_0116: Simulink patterns for logical constructs with logical blocks`
- `db_0117: Simulink patterns for vector signals`
- `jc_0351: Methods of initialization`
- `jc_0111: Direction of Subsystem`

The preceding guidelines illustrate sample patterns used in Simulink diagrams. As such, the patterns normally would be part of a much larger Simulink diagram.

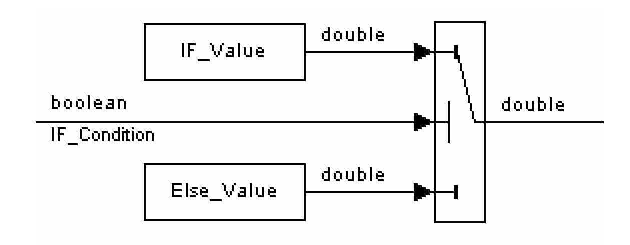
Some of the preceding guidelines refer to basic blocks. For an explanation of the meaning and some examples, see "Basic Blocks" on page D-2.

na_0012: Use of Switch vs. If-Then-Else Action Subsystem

ID: Title	na_0012: Use of Switch vs. If-Then-Else Action Subsystem
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None

Description

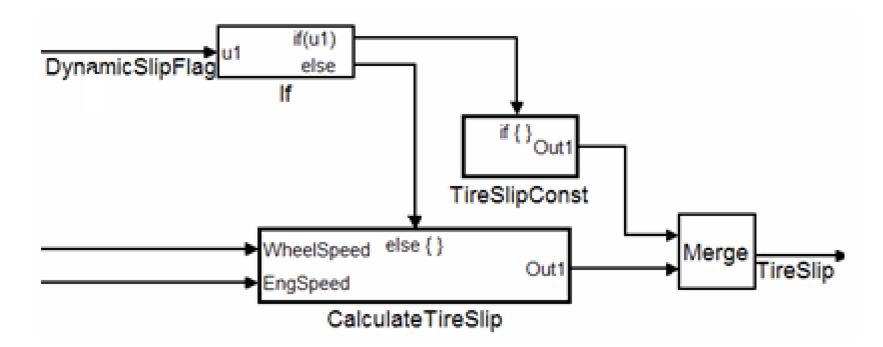
The **Switch** block should be used for modeling simple *if-then-else* structures, if the associated *then* and *else* actions involve only the assignment of constant values.



The **if-then-else** action subsystem construct:

- Should be used for modeling *if-then-else* structures, if the associated *then* and/or *else* actions require complicated computations. This maximizes simulation efficiency and the efficiency of generated code. (Note that even a basic block, for example a table lookup, may require fairly complicated computations.)

na_0012: Use of Switch vs. If-Then-Else Action Subsystem



- Must be used for modeling *if-then-else* structures, if the purpose of the construct is to avoid an undesirable numerical computation, such as division by zero.
- Should be used for modeling *if-then-else* structures, if the explicit or implied *then* or the *else* action is just to hold the associated output values.

In other cases, the degree of complexity of the *then* and/or *else* action computations and the intelligence of the Simulink simulation and code generation engines determine the appropriate construct.

These statements also apply to more complicated nested and cascaded *if-then-else* structures and *case* structure implementations.

Rationale

- Readability
- Workflow

Last Changed

V2.0

Model Advisor Check

Not applicable

db_0114: Simulink patterns for If-then-else-if constructs

ID: Title db_0114: Simulink patterns for If-then-else-if constructs

Priority Strongly recommended

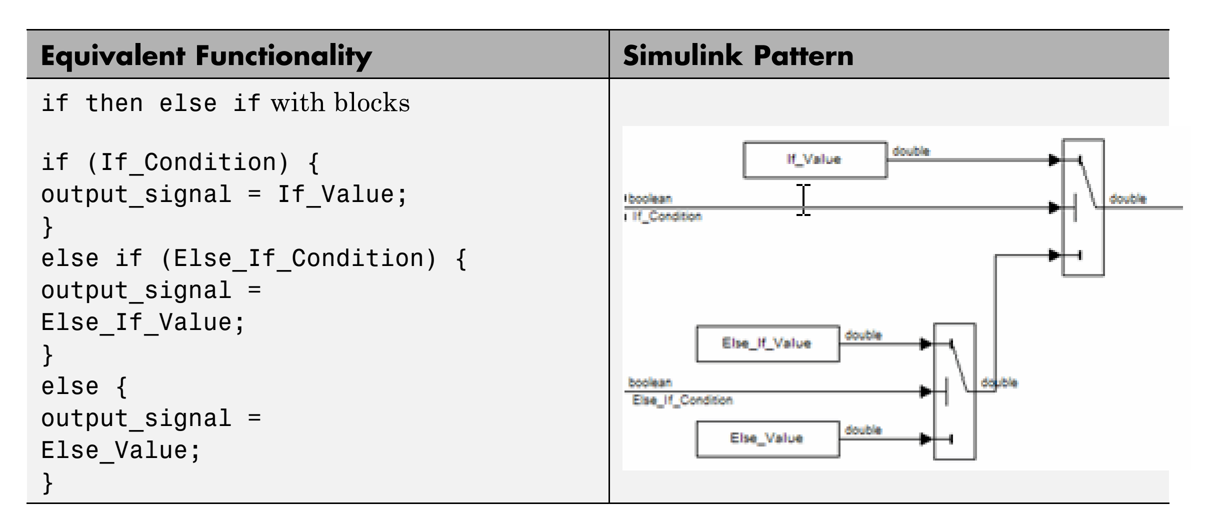
Scope MAAB

MATLAB Versions All

Prerequisites None

Description Use the following patterns for If-then-else-if constructs within a Simulink model:

Equivalent Functionality	Simulink Pattern

`if then else if` with blocks
 	
`if (If_Condition) {`
`output_signal = If_Value;`
`}`
`else if (Else_If_Condition) {`
`output_signal =`
`Else_If_Value;`
`}`
`else {`
`output_signal =`
`Else_Value;`
`}`
	

db_0114: Simulink patterns for If-then-else-if constructs

Equivalent Functionality	Simulink Pattern

`if then else if` with if/then/else subsystems
 	
`if(Fault_1_Active &`
`Fault_2_Active)`
`{`
`  ErrMsg = SaftyCrit;`
`}`
`else if (Fault_1_Active `|
`Fault_2_Active)`
`{`
`  ErrMsg = DriveWarn;`
`}`
`else`
`{`
`  ErrMsg = NoFaults;`
`}`	

Rationale

- Readability
- Workflow
- Code Generation

Last Changed

V2.0

Model Advisor Check

Not applicable

db_0115: Simulink patterns for case constructs

ID: Title db_0115: Simulink patterns for case constructs

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description Use the following patterns for case constructs within a Simulink model:

Equivalent Functionality	Simulink Pattern

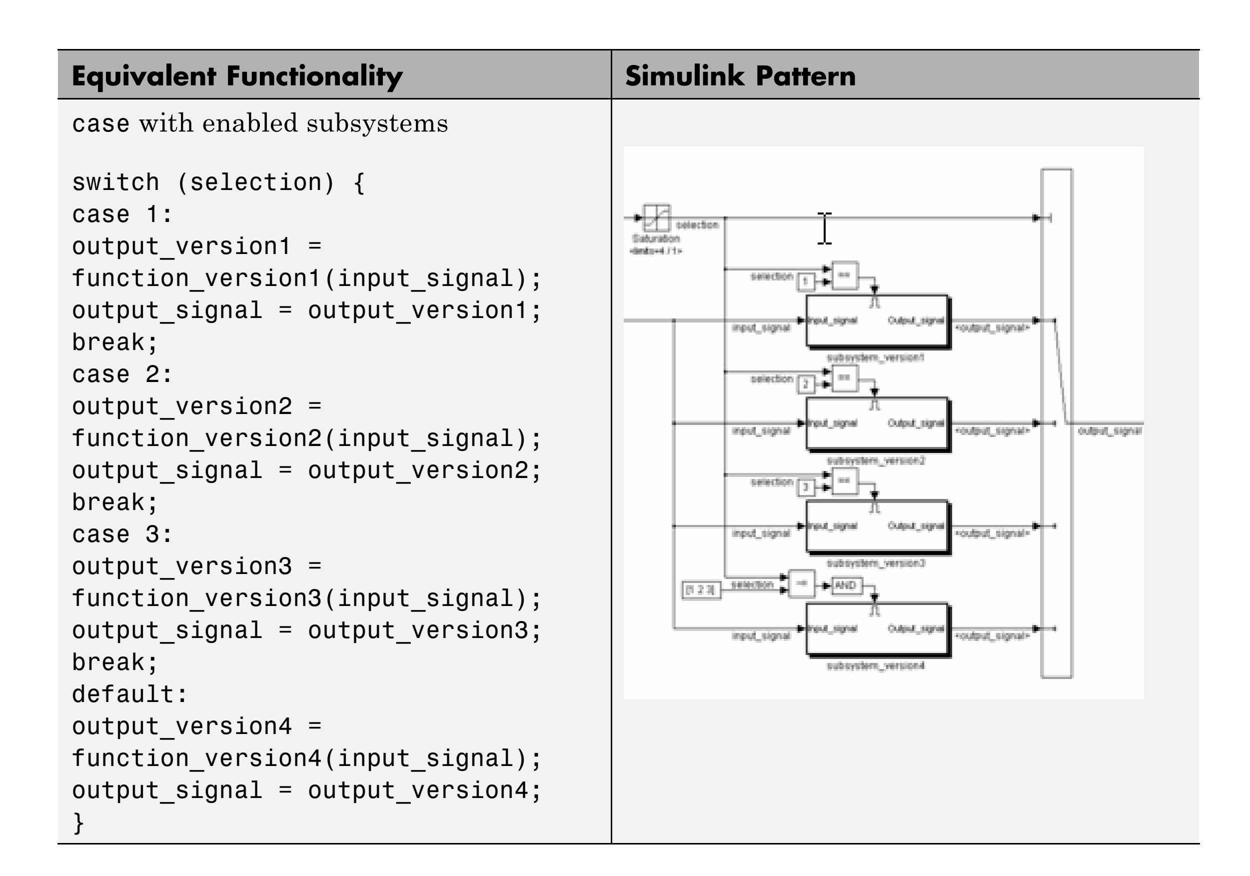
`case` with Switch Case block
 	
`switch (PRNDL_Enum)`
`{`
`case 1`
`  TqEstimate = ParkV;`
`  break;`
`case 2`
`  TqEstimae = RevV;`
`  break;`
`default`
`  TqEstimate = NeutralV;`
`  break;`	

Equivalent Functionality	Simulink Pattern
}	

case with subsystems
 	
output_version1 =
function_version1(input_signal);
output_version2 =
function_version2(input_signal);
output_version3 =
function_version3(input_signal);
output_version4 =
function_version4(input_signal);
switch (selection) {
case 1:
output_signal = output_version1;
break;
case 2:
output_signal = output_version2;
break;
case 3:
output_signal = output_version3;
break;
case 4:
output_signal = output_version4;
}	

db_0115: Simulink patterns for case constructs

Equivalent Functionality	Simulink Pattern

case with enabled subsystems
 	
`switch (selection) {`
`case 1:`
`output_version1 =`
`function_version1(input_signal);`
`output_signal = output_version1;`
`break;`
`case 2:`
`output_version2 =`
`function_version2(input_signal);`
`output_signal = output_version2;`
`break;`
`case 3:`
`output_version3 =`
`function_version3(input_signal);`
`output_signal = output_version3;`
`break;`
`default:`
`output_version4 =`
`function_version4(input_signal);`
`output_signal = output_version4;`
`}`
	

Rationale

- Readability
- Workflow
- Verification and Validation

Last Changed

V2.0

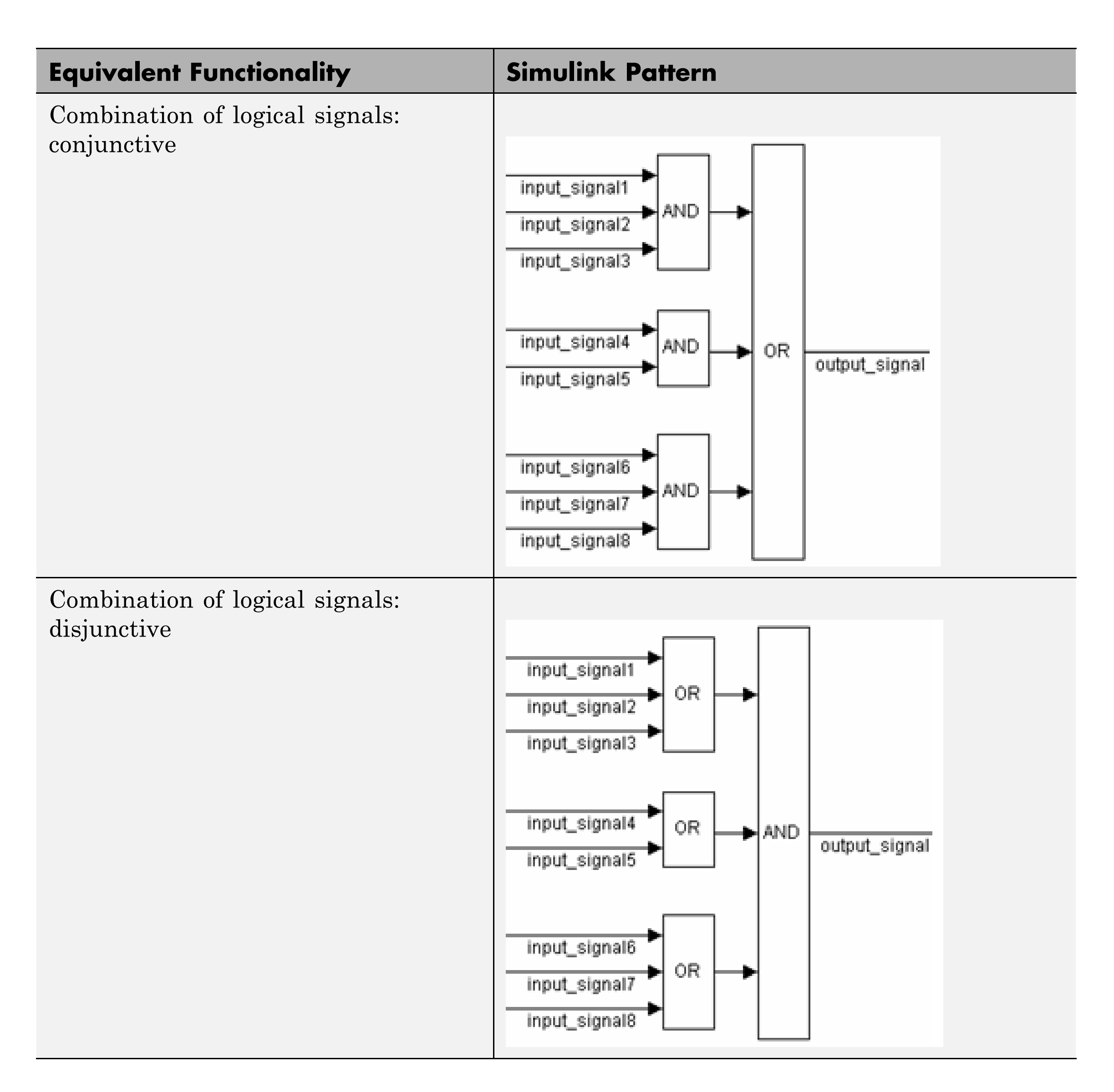
Model Advisor Check

Not applicable

db_0116: Simulink patterns for logical constructs with logical blocks

ID: Title	db_0116: Simulink patterns for logical constructs with logical blocks
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None
Description	Use the following patterns for logical combinations within Simulink:

db_0116: Simulink patterns for logical constructs with logical blocks



Equivalent Functionality	Simulink Pattern
Combination of logical signals: conjunctive	
Combination of logical signals: disjunctive	

Rationale

- Readability

- Workflow
- Verification and Validation

Last Changed

V1.0

Model Advisor Check

Not applicable

db_0117: Simulink patterns for vector signals

ID: Title db_0117: Simulink patterns for vector signals

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description Use the following patterns for vector signals within a Simulink model:

Equivalent Functionality	Simulink Pattern

Vector loop
 	
for (i=0;
i>input_vector_size; i++)
{
output_vector(i) =
input_vector(i) *
tunable_parameter_value;
}	

Vector loop
 	
for (i=0;
i>input_vector_size; i++)
{
output_vector(i) =
input_vector(i) *
tunable_parameter_vector(i);
}	

Equivalent Functionality	Simulink Pattern

Vector loop
 	
output_signal = 1;
for (i=0;
i>input_vector_size; i++)
{
output_signal =
output_signal *
input_vector(i);
}
	input_vector
	output_signal
	Product

Vector loop
 	
output_signal = 1;
for (i=0;
i>input_vector_size; i++)
{
output_signal =
output_signal /
input_vector(i);
}
	input_vector
	output_signal
	Product

Vector loop
 	
for (i=0;
i>input_vector_size; i++)
{
output_vector(i) =
input_vector(i) +
tunable_parameter_value;
}
	input_vector
	tunable_parameter_value
	Constant
	output_vector
	Sum

Equivalent Functionality	Simulink Pattern

Vector loop
 	
`for (i=0;`
`i>input_vector_size; i++)`
`{`
`output_vector(i) =`
`input_vector(i) +`
`tunable_parameter_vector(i);`
`}`
	input_vector
	tunable_parameter_vector
	Constant
	output_vector
	Sum

Vector loop:
 	
`output_signal = 0;`
`for (i=0;`
`i>input_vector_size; i++)`
`{`
`output_signal =`
`output_signal +`
`input_vector(i);`
`}`
	input_vector
	output_signal
	Sum

Vector loop:
 	
`output_signal = 0;`
`for (i=0;`
`i>input_vector_size; i++)`
`{`
`output_signal =`
`output_signal -`
`input_vector(i);`
`}`
	input_vector
	output_signal
	Sum

db_0117: Simulink patterns for vector signals

Equivalent Functionality	Simulink Pattern
Minimum or maximum of a signal or a vector over time:	input_signal; min; MinMax; 1/z; Unit_Delay; output_signal_min; input_vector; max; MinMax; 1/z; Unit_Delay; output_vector_max
Change event of a signal or a vector:	input_signal; 1/z; Unit_Delay; ~=; Relational Operator; output_signal_change; input_vector; 1/z; Unit_Delay; ~=; Relational Operator; output_vector_change; input_vector; 1/z; Unit_Delay; ~=; Relational Operator; OR; Logical Operator; output_vector_change

Rationale

- Readability
- Workflow
- Verification and Validation
- Code Generation

Last Changed

V1.0

Model Advisor Check

Not applicable

jc_0351: Methods of initialization

ID: Title

jc_0351: Methods of initialization

Priority

Recommended

Scope

MAAB

MATLAB Versions

All

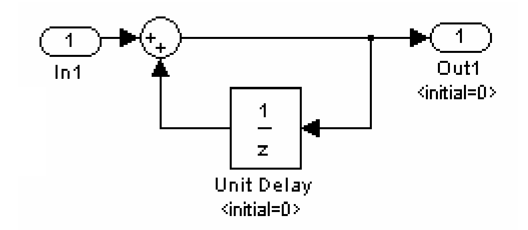
Prerequisites

`db_0140:  Display of basic block parameters`

Description

Simple Initialization

- Blocks such as Unit Delay, which have an initial value field, can be used to set simple initial values.
- To determine if the initial value needs to be displayed, see MAAB Guideline `db_0140:  Display of basic block parameters`.



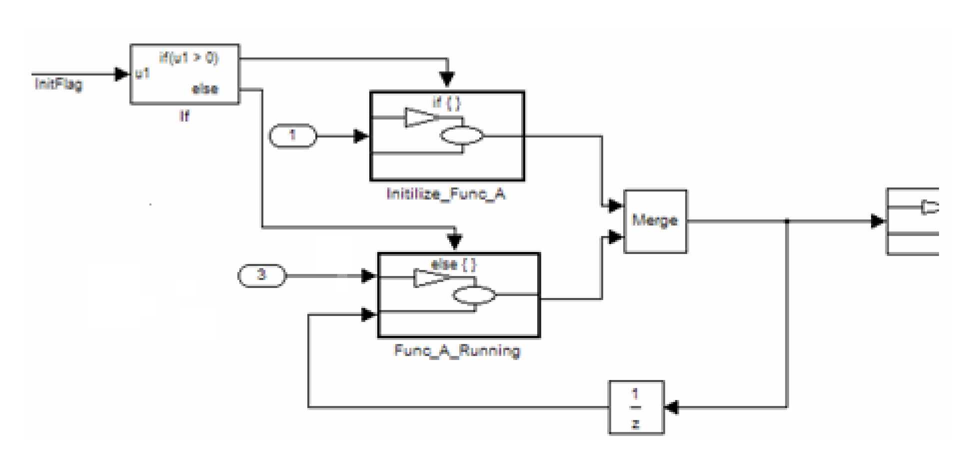
Example

Initialization that Requires Computation

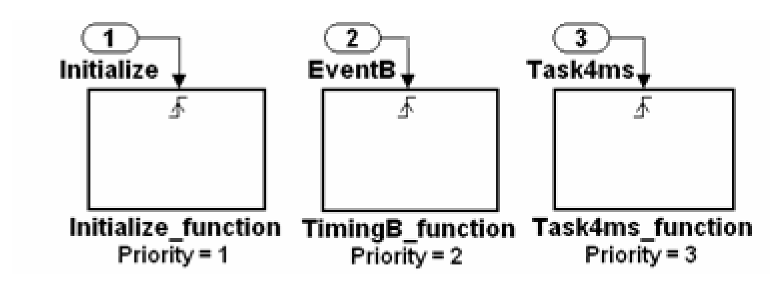
The following rules apply for complex initialization:

- The initialization should be performed in a separate subsystem.
- The initialization subsystem should have a name that indicates that initialization is performed by the subsystem.

Complex initialization may be done at a local level (Example A), at a global level (Example B), or a combination of local and global.



Example A



Example B

Rationale

Workflow

Last Changed

V2.0

Model Advisor Check

Not applicable

jc_0111: Direction of Subsystem

ID: Title jc_0111: Direction of Subsystem

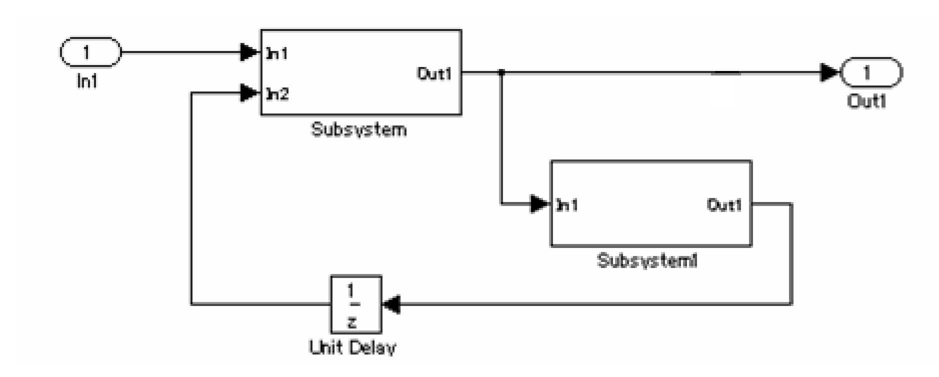
Priority Strongly recommended

Scope J-MAAB

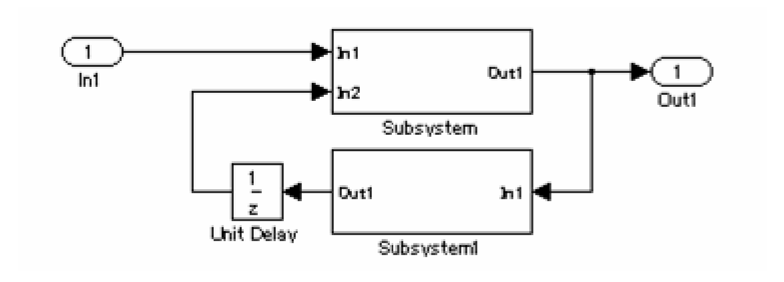
MATLAB Versions All

Prerequisites None

Description Subsystem must not be reversed.



Correct



Incorrect

jc_0111: Direction of Subsystem

Rationale

Readability

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Simulink > "Check orientation of Subsystem blocks"

jc_0111: Direction of Subsystem

6

Stateflow

Chart Appearance

- db_0123: Stateflow port names
- db_0129: Stateflow transition appearance
- db_0137: States in state machines
- db_0133: Use of patterns for Flowcharts
- db_0132: Transitions in Flowcharts
- jc_0501: Format of entries in a State block
- jc_0511: Setting the return value from a graphical function
- jc_0531: Placement of the default transition
- jc_0521: Use of the return value from graphical functions

db_0123: Stateflow port names

ID: Title db_0123: Stateflow port names

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description The name of a Stateflow input or output should be the same as the corresponding signal.

Exception: Reusable Stateflow blocks may have different port names.

Rationale

- Readability
- Workflow

Last Changed V1.0

Model Advisor Check **By Task > Modeling Standards for MAAB > Stateflow > "Check for mismatches between names of Stateflow ports and associated signals"**

db_0129: Stateflow transition appearance

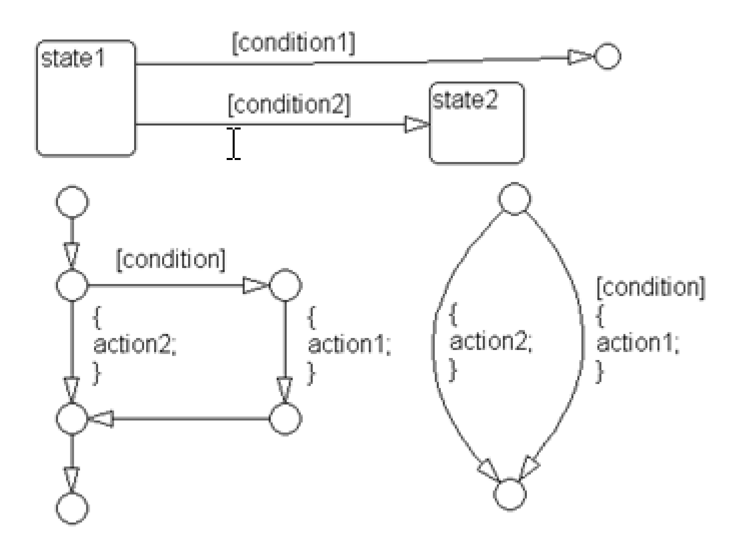
ID: Title	db_0129: Stateflow transition appearance
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None

Description

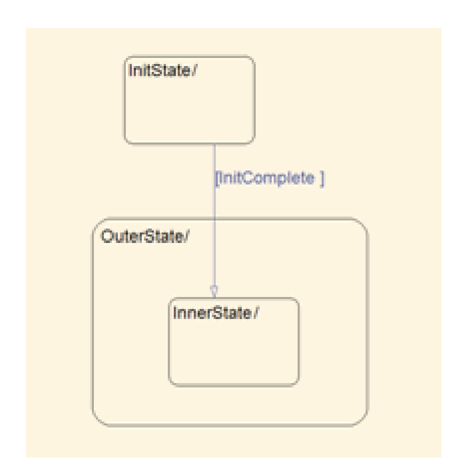
Transitions in Stateflow:

- Do not cross each other, if possible
- Are not drawn one upon the other
- Do not cross any states, junctions, or text fields
- Are allowed if transition is to an internal state

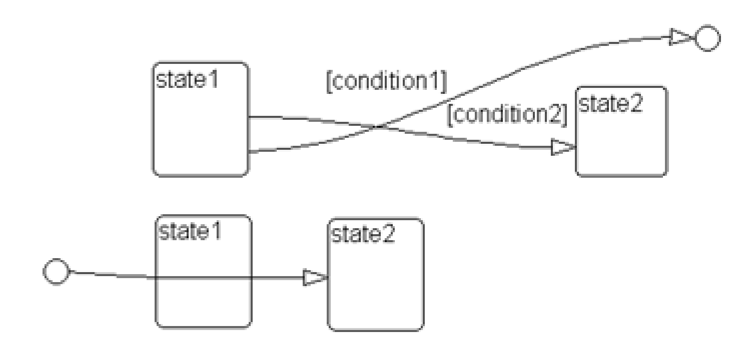
Transition labels may be visually associated to the corresponding transition.


state1
[condition1]
[condition2]
state2
[condition]
{
action2;
}
{
action1;
}
{
action2;
}
[condition]
{
action1;
}

Correct


InitState/
[InitComplete]
OuterState/
InnerState/

db_0129: Stateflow transition appearance



Incorrect

Rationale

- Readability
- Workflow

Last Changed

V2.0

Model Advisor Check

Not applicable

db_0137: States in state machines

ID: Title	db_0137: States in state machines
Priority	Mandatory
Scope	MAAB
MATLAB Versions	All
Prerequisites	`db_0149:  Flowchart patterns for condition actions`
Description	In state machines: • At least two exclusive states exist. • A state cannot have only one substate. • The initial state of a hierarchical level with exclusive states is clearly defined by a default transition.
Rationale	• Readability • Workflow • Verification and Validation
Last Changed	V2.0
Model Advisor Check	**By Task > Modeling Standards for MAAB > Sstateflow > "Check for exclusive and default states and substate correctness"**

db_0133: Use of patterns for Flowcharts

ID: Title	db_0133: Use of patterns for Flowcharts
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None
Description	A Flowchart is built with the help of Flowchart patterns (for example, `if-then-else`, `for` loop, and so on): • The data flow is oriented from the top to the bottom. • • Patterns are connected with empty transitions.
Rationale	• Readability • Workflow • Verification and Validation
Last Changed	V1.0
Model Advisor Check	Not applicable

db_0132: Transitions in Flowcharts

ID: Title db_0132: Transitions in Flowcharts

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

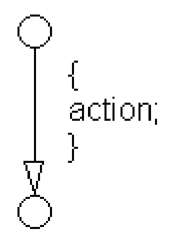
Prerequisites None

Description The following rules apply to transitions in Flowcharts:

- Conditions are drawn on the horizontal.
- Actions are drawn on the vertical.
- Loop constructs are intentional exceptions to this rule.
- Transitions have a condition, a condition action, or an empty transition.

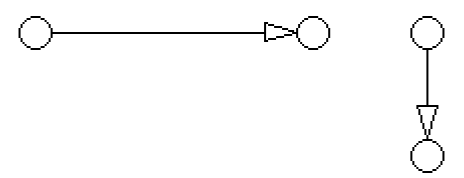


Transition with Condition



Transition with Condition Action

db_0132: Transitions in Flowcharts

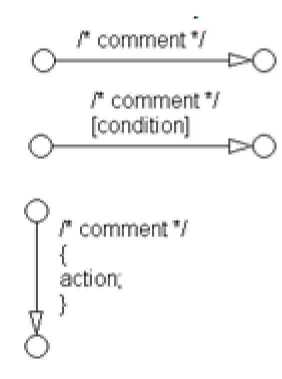


Empty Transition

Transition actions are not used in Flowcharts. Transition actions are only valid when used in transitions between states in a state machine, otherwise they are not activated because of the inherent dependency on a valid state to state transition to activate them.

Transition Action

At every junction, except for the last junction of a flow diagram, exactly one unconditional transition begins. Every decision point (junction) must have a default path.



Transitions with Comments

Rationale

- Readability
- Workflow
- Verification and Validation

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Stateflow > "Check Transition orientations in flowcharts"

jc_0501: Format of entries in a State block

ID: Title	jc_0501: Format of entries in a State block
Priority	Recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None

Description

A new line should be:

- Started after the entry (en), during (du), and exit (ex) statements.
- Started after the completion of an assignment statement ";".

```
State
en:
entry_value=1;
during_value=0;
du:
entry_value=0;
during_value=1;
ex:
exit_value=1;
```

Correct

```
State
en:entry_value=1;
during_value=0;
du:entry_value=0;
during_value=1;
ex:exit_value=2;
```

Incorrect

Failed to start a new line after `en`, `du`, and `ex`.

```
State
en:entry_value=1;during_value=0;du:entry_value=0;
during_value=1;ex:exit_value=2;
```

Incorrect

Failed to start a new line after the completion of an assignment statement ";".

Rationale

Readability

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Stateflow > "Check entry formatting in State blocks in Stateflow charts"

jc_0511: Setting the return value from a graphical function

ID: Title jc_0511: Setting the return value from a graphical function

Priority Mandatory

Scope J-MAAB

MATLAB Versions All

Prerequisites None

Description The return value from a graphical function must be set in only one place.

Correct

Return value A is set in one place.

jc_0511: Setting the return value from a graphical function

Incorrect

Return value A is set in multiple places.

Rationale

- Workflow
- Code Generation

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Stateflow > "Check return value assignments of graphical functions in Stateflow charts"

jc_0531: Placement of the default transition

ID: Title
jc_0531: Placement of the default transition

Priority
Recommended

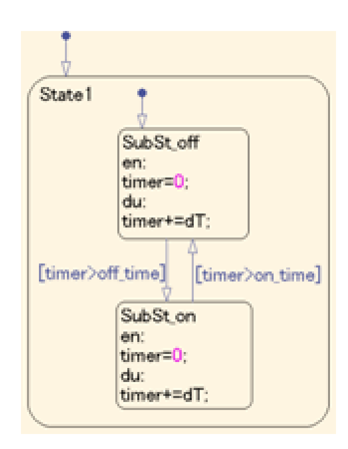
Scope
J-MAAB

MATLAB Versions
All

Prerequisites
None

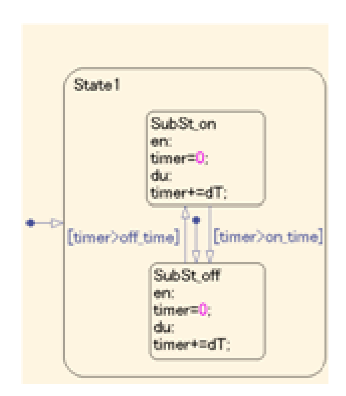
Description

- Default transition is connected at the top of the state.
- The destination state of the default transition is put above the other states in the same hierarchy.



Correct

- The default transition is connected at the top of the state.
- The destination state of the default transition is put above the other states in the same hierarchy.



Incorrect

- Default transition is connected at the side of the state (State 1).
- The destination state of the default transition is lower than the other states in the same hierarchy (`SubSt_off`).

Rationale

Readability

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Stateflow > "Check default transition placement in Stateflow charts"

jc_0521: Use of the return value from graphical functions

ID: Title jc_0521: Use of the return value from graphical functions

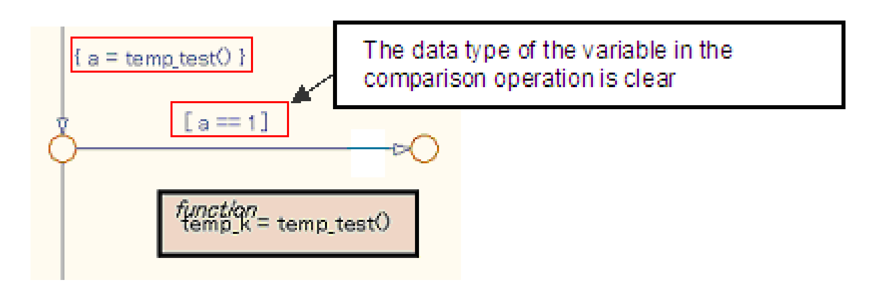
Priority Recommended

Scope J-MAAB

MATLAB Versions All

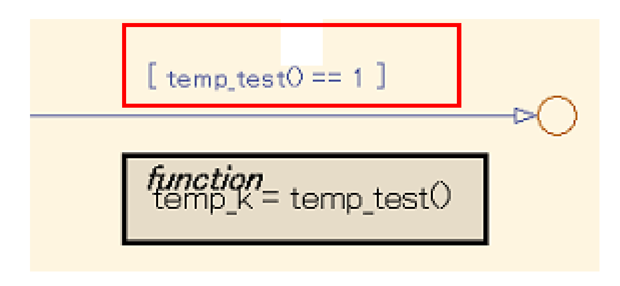
Prerequisites None

Description The return value from a graphical function should not be used directly in a comparison operation.



Correct

An intermediate variable is used in the conditional expression after the assignment of the return value from the function `temp_test` to the intermediate variable `a`.



Incorrect

Return value of the function `temp_test` is used in the conditional expression.

Rationale

Readability

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Stateflow > "Check usage of return values from a graphical function in Stateflow charts"

jc_0521: Use of the return value from graphical functions

Stateflow Data and Operations

- na_0001: Bitwise Stateflow operators
- jc_0451: Use of unary minus on unsigned integers in Stateflow
- na_0013: Comparison operation in Stateflow
- db_0122: Stateflow and Simulink interface signals and parameters
- db_0125: Scope of internal signals and local auxiliary variables
- jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow
- jc_0491: Reuse of variables within a single Stateflow scope
- jc_0541: Use of tunable parameters in Stateflow
- db_0127: MATLAB commands in Stateflow
- jm_0011: Pointers in Stateflow

na_0001: Bitwise Stateflow operators

ID: Title na_0001: Bitwise Stateflow operators

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description The bitwise Stateflow operators (&, |, and ^) should not be used in Stateflow charts unless you want bitwise operations.

To enable bitwise operations,

1. Select **File > Chart Properties** .
2. Select **Enable C-bit operations**.

Correct

Use && and || for Boolean operation.

Use & and | for bit operation.

Incorrect

Use & and | for Boolean operation.

Rationale

- Simulation
- Code Generation

Last Changed V2.0

Model Advisor Check **By Task > Modeling Standards for MAAB > Stateflow > "Check for bitwise operations in Stateflow charts"**

jc_0451: Use of unary minus on unsigned integers in Stateflow

ID: Title jc_0451: Use of unary minus on unsigned integers in Stateflow

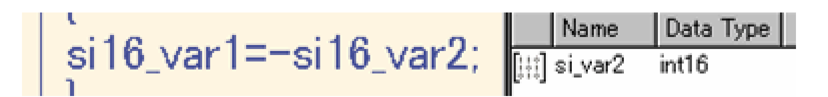
Priority Recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description Do not perform unary minus on unsigned integers.



Correct

ui16_var1=-ui16_var2;

Name	Data Type
ui_var2	uint16

Incorrect

Rationale

- Readability
- Workflow
- Code Generation

Last Changed V2.0

Model Advisor Check **By Task > Modeling Standards for MAAB > Stateflow > “Check for unary minus operations on unsigned integers in Stateflow charts”**

na_0013: Comparison operation in Stateflow

ID: Title

na_0013: Comparison operation in Stateflow

Priority

Recommended

Scope

MAAB

MATLAB Versions

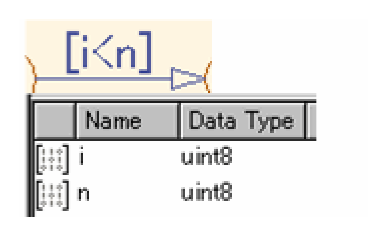
All

Prerequisites

None

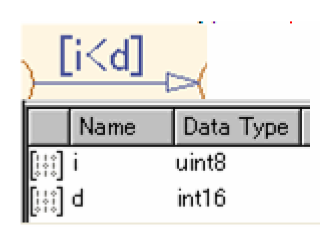
Description

- Comparisons should be made only between variables of the same data type.
- If comparisons are made between variables of different data types, the variables need to be explicitly type cast to matching data types.



Correct

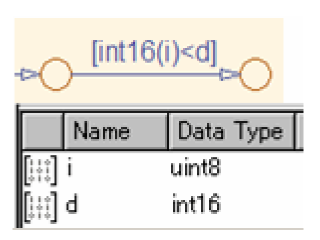
Same data type in “i” and “n”



Incorrect

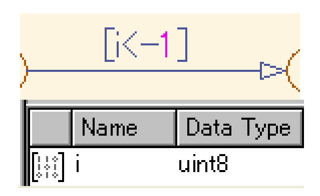
Different data type in “i” and “d”

na_0013: Comparison operation in Stateflow



Correct

Do not make comparisons between unsigned integers and negative numbers.



Incorrect

Rationale

- Workflow
- Code Generation

Last Changed

V2.1

Model Advisor Check

By Task > Modeling Standards for MAAB > Stateflow > “Check for comparison operations in Stateflow charts”

db_0122: Stateflow and Simulink interface signals and parameters

ID: Title

db_0122: Stateflow and Simulink interface signals and parameters

Priority

Strongly recommended

Scope

MAAB

MATLAB Versions

All

Prerequisites

None

Description

A Chart uses strong data typing with Simulink and requires that you select the **Use Strong Data Typing with Simulink I/O** parameter.

Rationale

- Readability
- Workflow
- Verification and Validation

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Stateflow > "Check for Strong Data Typing with Simulink I/O"

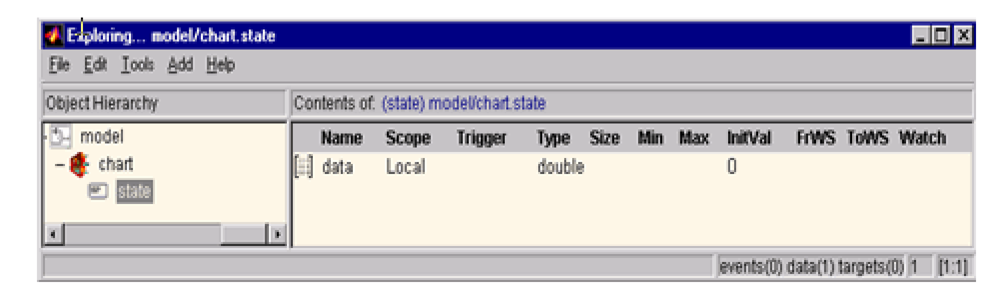
db_0125: Scope of internal signals and local auxiliary variables

ID: Title	db_0125: Scope of internal signals and local auxiliary variables
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None

Description

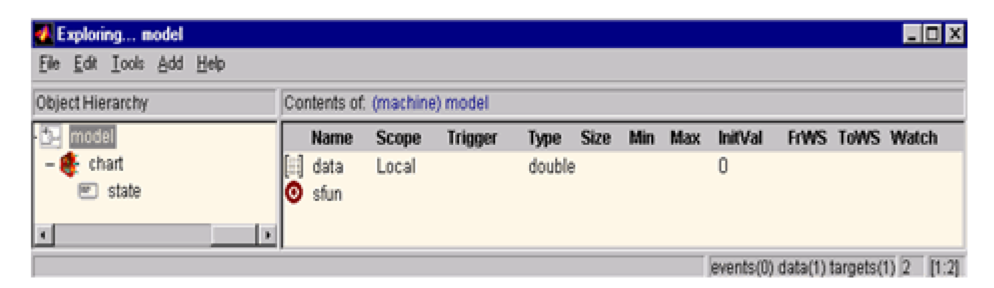
Internal signals and local auxiliary variables are "Local data" in Stateflow:

- All local data of a Stateflow block must be defined on the chart level or below the Object Hierarchy.
- No local variables may exist on the machine level (that is, no interaction should occur between local data in different charts).
- Parameters and constants are allowed at the machine level.



Correct

db_0125: Scope of internal signals and local auxiliary variables



Incorrect

Rationale

- Readability
- Workflow
- Verification and Validation

Last Changed

V2.0

Model Advisor Check

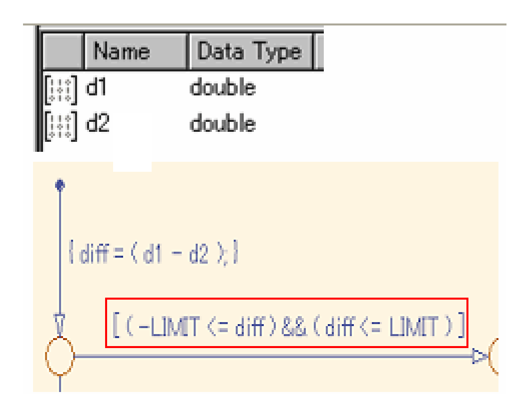
By Task > Modeling Standards for MAAB > Stateflow > "Check for Strong Data Typing with Simulink I/O"

jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow

ID: Title	jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow
Priority	Recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	None

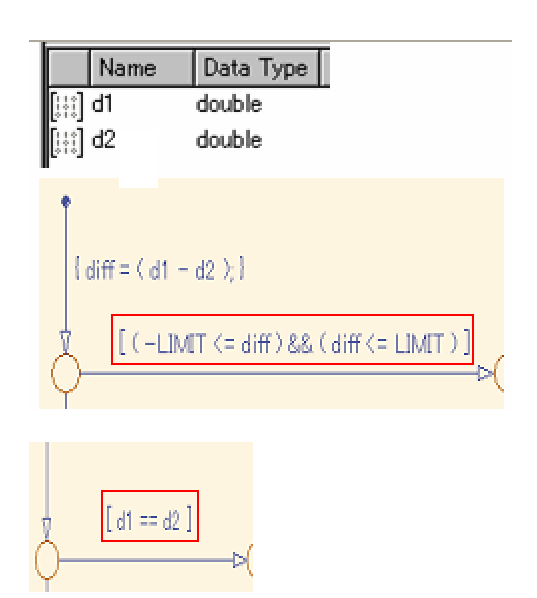
Description

- Do not use hard equality comparisons (`Var1 == Var2`) with two floating-point numbers.
- If a hard comparison is required, a margin of error should be defined and used in the comparison (LIMIT, in the example).
- Hard equality comparisons may be done between two integer data types.



Correct

jc_0481: Use of hard equality comparisons for floating point numbers in Stateflow



Incorrect

Rationale

- Workflow
- Verification and Validation
- Code Generation

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Stateflow > "Check for equality operations between floating-point expressions in Stateflow charts"

jc_0491: Reuse of variables within a single Stateflow scope

ID: Title jc_0491: Reuse of variables within a single Stateflow scope

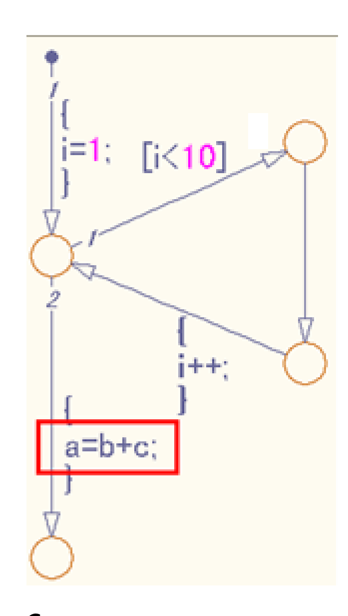
Priority Recommended

Scope MAAB

MATLAB Versions All

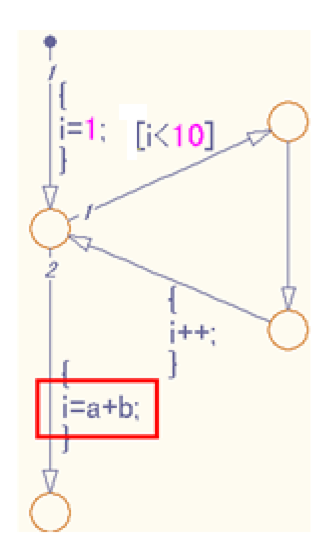
Prerequisites None

Description The same variable should not have multiple meanings (usages) within a single Stateflow scope.



Correct

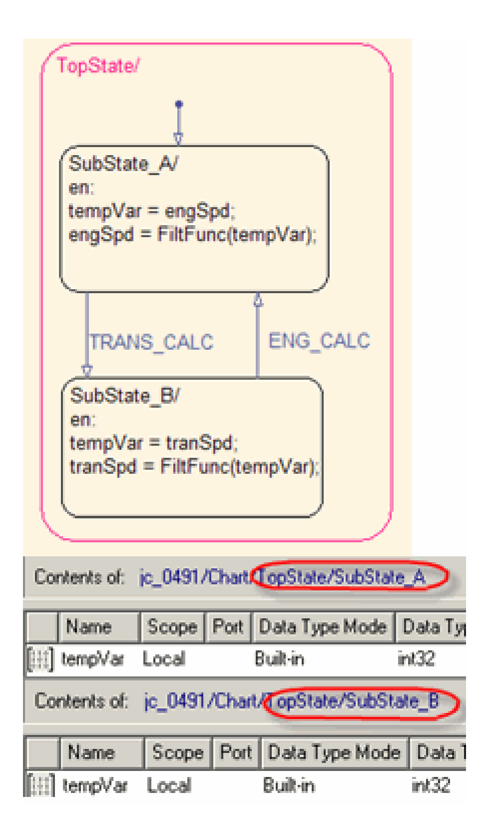
Variable of loop counter must not be used other than loop counter.



Incorrect

The meaning of the variable `i` changes from the index of the loop counter to the sum of `a+b`

jc_0491: Reuse of variables within a single Stateflow scope



Correct

`tempVar` is defined as local scope in both `SubState_A` and `SubState_B`.

Rationale

- Readability
- Workflow
- Code Generation

Last Changed

V2.0

jc_0491: Reuse of variables within a single Stateflow scope

Model Advisor Check

Not applicable

jc_0541: Use of tunable parameters in Stateflow

ID: Title jc_0541: Use of tunable parameters in Stateflow

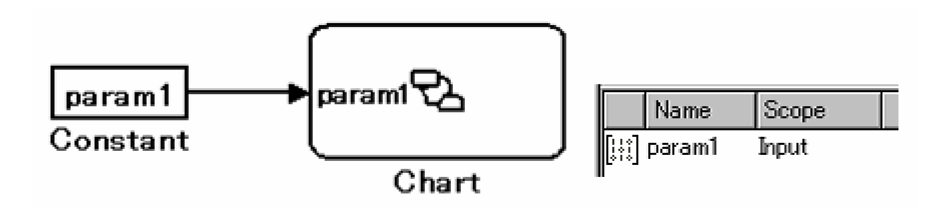
Priority Strongly recommended

Scope MAAB

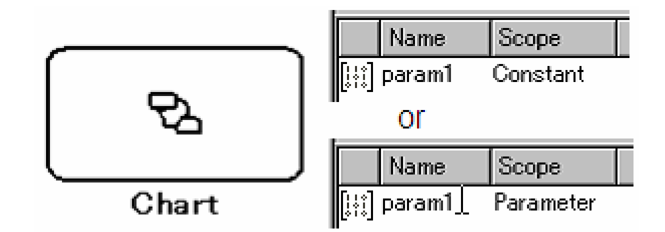
MATLAB Versions All

Prerequisites None

Description Tunable parameters should be included in a Chart as inputs from the Simulink model.



Correct



Incorrect

Rationale

- Readability
- Workflow
- Code Generation

jc_0541: Use of tunable parameters in Stateflow

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Stateflow > "Check for tunable parameters in Stateflow charts"

db_0127: MATLAB commands in Stateflow

ID: Title db_0127: MATLAB commands in Stateflow

Priority Mandatory

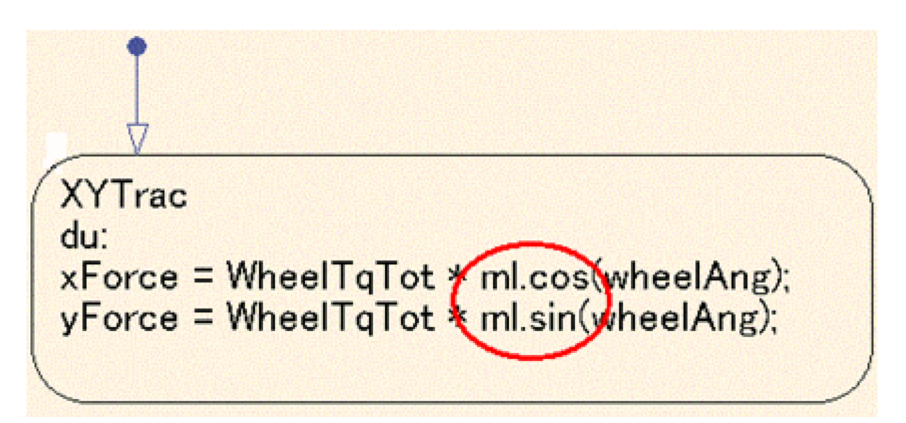
Scope MAAB

MATLAB Versions All

Prerequisites None

Description The following rules apply to logic in Stateflow:

- MATLAB functions are not used.
- MATLAB instructions are not used.
- MATLAB operators are not used.
- Project-specific MATLAB functions are not used.



Incorrect

Rationale

- Readability
- Workflow

- Verification and Validation
- Code Generation

Last Changed

V2.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Stateflow > “Check for MATLAB expressions in Stateflow charts”

jm_0011: Pointers in Stateflow

ID: Title jm_0011: Pointers in Stateflow

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description In a Stateflow diagram, pointers to custom code variables are not allowed.

Rationale

- Readability
- Workflow
- Verification and Validation
- Code Generation

Last Changed V1.0

Model Advisor Check **By Task > Modeling Standards for MAAB > Stateflow > "Check for pointers in Stateflow charts"**

Events

- `db_0126: Scope of events`
- `jm_0012: Event broadcasts`

db_0126: Scope of events

ID: Title db_0126: Scope of events

Priority Mandatory

Scope MAAB

MATLAB Versions All

Prerequisites None

Description The following rules apply to events in Stateflow:

- All events of a Chart must be defined on the chart level or lower.
- There is no event on the machine level (i.e. there is no interaction with local events between different charts).

Specifics

Rationale

- Readability
- Workflow
- Verification and Validation

Last Changed V2.0

Model Advisor Check **By Task > Modeling Standards for MAAB > Stateflow > "Check whether Stateflow events are defined at the chart level or below"**

ID: Title jm_0012: Event broadcasts

Priority Strongly recommended

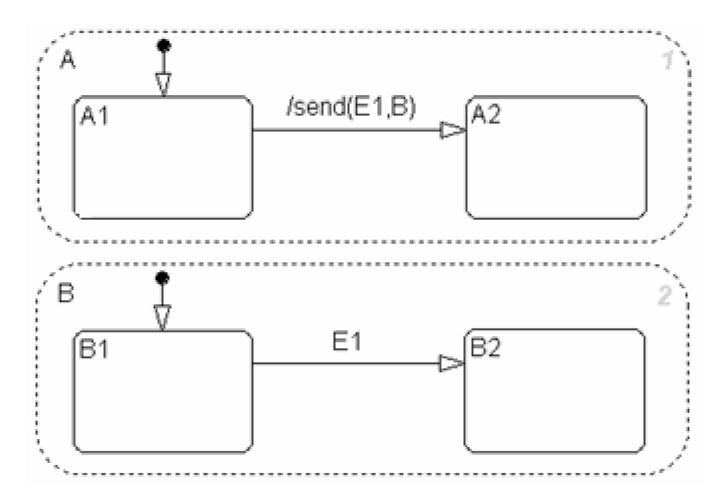
Scope MAAB

MATLAB Versions All

Prerequisites `db_0126: Scope of events`

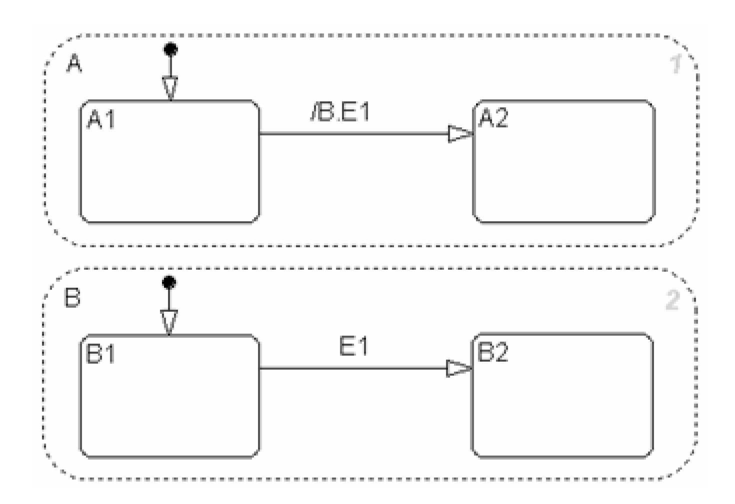
Description The following rules apply to event broadcasts in Stateflow:

- Directed event broadcasts are the only type of event broadcasts allowed.
- The send syntax or qualified event names are used to direct the event to a particular state.
- Multiple send statements should be used to direct an event to more than one state.



Example Using Send Syntax

jm_0012: Event broadcasts



Example Using Qualified Event Names

Rationale

- Readability
- Workflow
- Verification and Validation
- Code Generation

Last Changed

V1.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Stateflow > "Check for event broadcasts in Stateflow charts"

Statechart Patterns

- `db_0150: State machine patterns for conditions`
- `db_0151: State machine patterns for transition actions`

db_0150: State machine patterns for conditions

ID: Title
db_0150: State machine patterns for conditions

Priority
Strongly recommended

Scope
MAAB

MATLAB Versions
All

Prerequisites
None

Description
The following patterns are used for conditions within Stateflow state machines:

Equivalent Functionality	State Machine Pattern
One condition: `(condition)`	A —[condition]→ B

Equivalent Functionality	State Machine Pattern

Up to three conditions, short form:
(The use of different logical operators in this form is not allowed. Use subconditions instead.)
`(condition1 && condition2)`
`(condition1 || condition2)`
	A [condition1 && condition2] B
	A [condition1 || condition2] B

Two or more conditions, multiline form:
A subcondition is a set of logical operations, all of the same type, enclosed in parentheses.
(The use of different operators in this form is not allowed. Use subconditions instead.)
`(condition1 ...`
`&& condition2 ...`
`&& condition3)`
`(condition1 ...`
`|| condition2 ...`
`|| condition3)`
	A [condition1 ... && condition2 ... && condition3] B
	A [condition1 ... || condition2 ... || condition3] B

Rationale

- Readability
- Workflow

db_0150: State machine patterns for conditions

- Verification and Validation

Last Changed

V2.0

Model Advisor Check

Not applicable

db_0151: State machine patterns for transition actions

ID: Title

db_0151: State machine patterns for transition actions

Priority

Strongly recommended

Scope

MAAB

MATLAB Versions

All

Prerequisites

None

Description

The following patterns are used for transition actions within Stateflow state machines:

Equivalent Functionality	State Machine Pattern
One transition action: `action;`	A —/action;→ B

Two or more transition actions, multiline form:
 	
(Two or more transition actions in one line are not allowed.)
 	
`action1;`
`action2;`
`action3;`
	A —/action1; action2; action3;→ B

db_0151: State machine patterns for transition actions

Rationale

- Readability
- Workflow
- Verification and Validation

Last Changed

V1.0

Model Advisor Check

By Task > Modeling Standards for MAAB > Stateflow > "Check transition actions in Stateflow charts"

Flowchart Patterns

- `db_0148: Flowchart patterns for conditions`
- `db_0149: Flowchart patterns for condition actions`
- `db_0134: Flowchart patterns for If constructs`
- `db_0159: Flowchart patterns for case constructs`
- `db_0135: Flowchart patterns for loop constructs`

The preceding guidelines illustrate sample patterns used in flow charts. As such, they would normally be part of a much larger Stateflow diagram.

db_0148: Flowchart patterns for conditions

ID: Title db_0148: Flowchart patterns for conditions

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description Use the following patterns for conditions within Stateflow Flowcharts:

Equivalent Functionality	Flowchart Pattern
One condition: `[condition]`	[condition] /* comment */ [condition]

Up to three conditions, short form:
(The use of different logical operators in this form is not allowed. Use subconditions instead.)
 	
`[condition1`
`&& condition2`
`&& condition3]`
`[condition1`
`|| condition2`
`|| condition3]`
	[condition1 && condition2 && condition3]
	[condition1 || condition2 || condition3]

Equivalent Functionality	Flowchart Pattern

Two or more conditions, multiline form:
(The use of different logical operators in this form is not allowed. Use subconditions instead.)
`[condition1 ...`
`&& condition2 ...`
`&& condition3]`
`[condition1 ...`
`|| condition2 ...`
`|| condition3]`
	[condition1 ...
	&& condition2 ...
	&& condition3]
	[condition1 ...
	|| condition2 ...
	|| condition3]

Conditions with subconditions:
(The use of different logical operators to connect subconditions is not allowed. The use of brackets is mandatory.)
`[(condition1a`
`|| condition1b) ...`
`&& (condition2a`
`|| condition2b) ...`
`&& (condition3)]`
`[(condition1a`
`&& condition1b) ...`
`|| (condition2a`
`&& condition2b) ...`
`|| (condition3)]`
	[(condition1a || condition1b) ...
	&& (condition2a || condition2b) ...
	&& condition3]
	[(condition1a && condition1b) ...
	|| (condition2a && condition2b) ...
	|| condition3]

db_0148: Flowchart patterns for conditions

Equivalent Functionality	Flowchart Pattern

Conditions that are visually separated:
(This form may be combined with the preceding patterns.)
`[condition1 && condition2]`
`[condition1 || condition2]`
	[condition1] [condition2] [condition1] [condition2]

Rationale

- Readability
- Workflow
- Verification and Validation

Last Changed

V2.0

Model Advisor Check

Not applicable

db_0149: Flowchart patterns for condition actions

ID: Title db_0149: Flowchart patterns for condition actions

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites None

Description You should use the following patterns for condition actions within Stateflow Flowcharts:

db_0149: Flowchart patterns for condition actions

Equivalent Functionality	Flowchart Pattern

One condition action:
 	
`action;`
	{
	action;
	}
	
	/* comment */
	{
	action;
	}

Two or more condition actions, multiline form:
 	
(Two or more condition actions in one line are not allowed.)
 	
`action1; ...`
`action2; ...`
`action3; ...`
	{
	action1;
	action2;
	action3;
	}

Condition actions, that are visually separated:
 	
(This form may be combined with the preceding patterns.)
 	
`action1a;`
`action1b;`
`action2;`
`action3;`
	{
	action1a;
	action1b;
	}
	{
	action2;
	}
	{
	action3;
	}

Rationale

- Readability
- Workflow
- Verification and Validation

Last Changed

V1.0

Model Advisor Check

Not applicable

db_0134: Flowchart patterns for If constructs

ID: Title db_0134: Flowchart patterns for If constructs

Priority Strongly recommended

Scope MAAB

MATLAB Versions All

Prerequisites `db_0148: Flowchart patterns for conditions`

`db_0149: Flowchart patterns for condition actions`

Description Use the following patterns for If constructs within Stateflow Flowcharts:

Equivalent Functionality	Flowchart Pattern

`if then`
 	
`if (condition){ action;`
`}`
	[condition]
	{
	action;
	}

Equivalent Functionality	Flowchart Pattern

`if then else`
 	
`if (condition){   action1;`
`}`
`else {`
`  action2;`
`}`	

`if then else if`
 	
`if (condition1){   action1;`
`}`
`else if (condition2) {  action2;`
`}`
`else if (condition3){`
`__action3;`
`}`
`else {`
`  action4;`
`}`	

db_0134: Flowchart patterns for If constructs

Equivalent Functionality	Flowchart Pattern

Cascade of `if then`
 	
`if (condition1){  action1;`
`  if (condition2){    action2;`
`    if (condition3){      action3;`
`    }`
`  }`
`}`	

Rationale

- Readability
- Workflow
- Verification and Validation

Last Changed

V1.0

Model Advisor Check

Not applicable

db_0159: Flowchart patterns for case constructs

ID: Title	db_0159: Flowchart patterns for case constructs
Priority	Strongly recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	`db_0148: Flowchart patterns for conditions` `db_0149: Flowchart patterns for condition actions`

db_0159: Flowchart patterns for case constructs

Description

Use the following patterns must be used for case constructs within Stateflow Flowcharts:

Equivalent Functionality	Flowchart Pattern

`case` with exclusive selection
 	
`selection = ...;`
`switch (selection)`
`{`
`case 1:`
`  action1;`
`break;`
`case 2:`
`  action2;`
`break;`
`case 3:`
`  action3;`
`break;`
`default:`
`  action4;`
`}`	

Equivalent Functionality	Flowchart Pattern

`case` with exclusive conditions
 	
`c1 = condition1;`
`c2 = condition2;`
`c3 = condition3;`
`if (c1 && !c2 && !c3)`
`{`
`action1;`
`}`
`elseif (!c1 && c2 && !c3)`
`{`
`action2;`
`}`
`elseif (!c1 && !c2 && c3)`
`{`
`action3;`
`}`
`else`
`{`
`action4;`
`}`	

Rationale

- Readability
- Workflow
- Verification and Validation

Last Changed

V1.0

db_0159: Flowchart patterns for case constructs

Model Advisor Check	Not applicable

db_0135: Flowchart patterns for loop constructs

ID: Title	db_0135: Flowchart patterns for loop constructs
Priority	Recommended
Scope	MAAB
MATLAB Versions	All
Prerequisites	`db_0148: Flowchart patterns for conditions` `db_0149: Flowchart patterns for condition actions`

db_0135: Flowchart patterns for loop constructs

Description

Use the following patterns to create Loops within Stateflow Flowcharts:

Equivalent Functionality	Flowchart Pattern

`for` loop
 	
`for (index=0;`
`index<number_of_loops;`
`index++)`
`{`
`action;`
`}`	

`while` loop
 	
`while (condition)`
`{`
`action;`
`}`	

`do while` loop
 	
`do`
`{`
`action;`
`}`
`while (condition);`	

db_0135: Flowchart patterns for loop constructs

Rationale

- Readability
- Workflow
- Verification and Validation

Last Changed

V1.0

Model Advisor Check

Not applicable

db_0135: Flowchart patterns for loop constructs

Recommendations for Automation Tools

These recommendations are for companies who develop tools that automate checking of the style guidelines. The MathWorks Automotive Advisory Board (MAAB) developed these recommendations for tool vendors who create tools developed with MathWorks tools that check models against these guidelines. To provide maximum information to potential users of the tools, the MAAB strongly recommends that tool vendors provide a compliance matrix that is easily accessible while the tool is running. This information should be available without a need to purchase the tool.

The compliance matrix should include the following information:

- Version of the guidelines that are checked – shall include the complete title, as found on the title page of this document.

  Include the MAAB Style Guidelines Title and Version document number.

- Table consisting of the following information for each guideline:
  - Guideline ID
  - Guideline title
  - Level of compliance
  - Detail

The guideline ID and title shall be exactly as included in this document. The level of compliance shall be one of the following:

Correction	The tool checks and automatically or semiautomatically corrects the noncompliance.
Check	The tool checks and flags noncompliance. It is the developer's responsibility to make the correction.
Partial	The tool checks part of the guideline. The detail section should clearly identify what is and what is not checked.
None	The tool does not check the guideline. The MAAB recommends that the vendor provide a recommendation of how to manually check guidelines that the tool does not check.

Guideline Writing

Guidelines with the following characteristics are easier to understand and use. At a minimum, when writing a new guideline, it should be

Understandable and unambiguous	A guideline's description should be precise, clearly worded, concise, and should define a characteristic of a model (or part of a model) that a checking tool can evaluate. Use the words "must," "shall," "should," and "may" carefully; they have distinct meanings that are important for model developers and model checkers (human and automated). It is helpful to the reader if the guideline author describes how the conforming state can be reached (for example, by selecting particular options or clicking a certain button). Examples, counterexamples, pictures, diagrams, and screen shots are also helpful and are encouraged. Minimize the allowable exceptions to a guideline; exceptions blur a guideline and make it harder to apply. If a guideline has many allowable exceptions, you may be trying to cover too many characteristics with one guideline. (See Minimal, following, for some solutions.)
Easy to find	
Minimal	A guideline should address only one model characteristic at a time. Guidelines should be atomic. For example, instead of writing a big guideline that addresses error prevention and readability at the same time, make

	two guidelines, one that addresses error prevention and one that addresses readability. If appropriate, make one guideline a prerequisite of the other. Also, big guidelines are more likely than small guidelines to require compromises for wide acceptance. Big guidelines may end up being weaker, less specific, and less beneficial. Small, focused guidelines are less likely to change due to compromise and easier adoption.

C

Flowchart Reference

Use the patterns that appear in this appendix for if-then-else-if constructs within Stateflow Flowcharts.

Straight Line Flow Chart Pattern	Curved Line Flow Chart Pattern
`if then`	

[condition]
{
action;
}
	[condition]
	{
	action;
	}

`if then else`	

[condition]
{
action2;
}
{
action1;
}
	{
	action2;
	}
	[condition]
	{
	action1;
	}

Straight Line Flow Chart Pattern	Curved Line Flow Chart Pattern
`if then else if`	
Cascade of `if then`	

The following patterns are used for case constructs within Stateflow Flowcharts:

Straight Line Flow Chart Pattern	Curved Line Flow Chart Pattern
case with exclusive selection	
{ selection = ...; } [selection == 1] [selection == 2] [selection == 3] { action4; } { action3; } { action2; } { action1; }	{ selection = ...; } { action4; } [selection == 3] { action3; } [selection == 2] { action2; } [selection == 1] { action1; }

Straight Line Flow Chart Pattern	**Curved Line Flow Chart Pattern**
`case` with exclusive conditions	

{
c1 = condition1;
c2 = condition2;
c3 = condition3;
}
[c1 && !c2 && !c3]
[!c1 && c2 && !c3]
[!c1 && !c2 && c3]
{
action4;
}
{
action3;
}
{
action2;
}
{
action1;
}
	{
	c1 = condition1;
	c2 = condition2;
	c3 = condition3;
	}
	{
	action4;
	}
	[!c1 && !c2 && c3]
	{
	action3;
	}
	[!c1 && c2 && !c3]
	{
	action2;
	}
	[c1 && !c2 && !c3]
	{
	action1;
	}

The following patterns are used for `for` loops within Stateflow Flowcharts:

Straight Line Flow Chart Pattern	Curved Line Flow Chart Pattern
`for` loop	

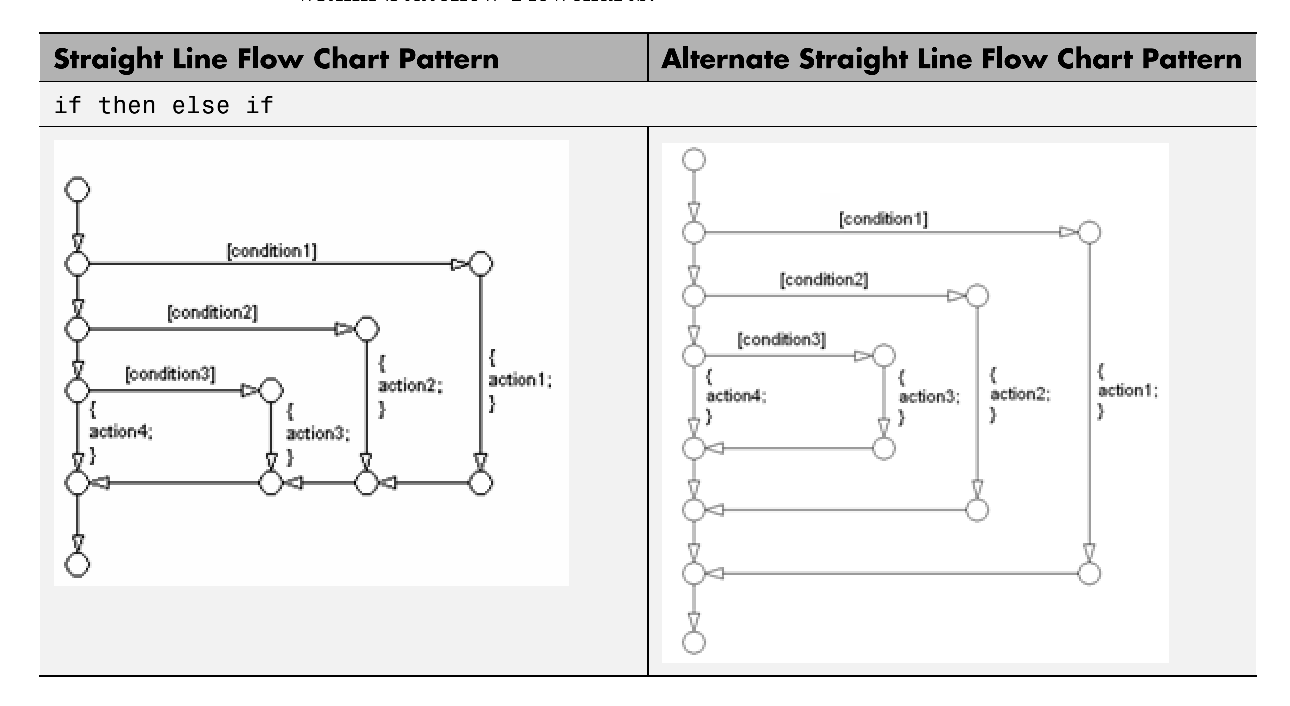
[index < number_of_loops]
{
index = 0;
}
{
action;
}
{
index++;
}
	{
	index = 0;
	}
	[index < number_of_loops]
	{
	action;
	index++;
	}

Straight Line Flow Chart Pattern	Curved Line Flow Chart Pattern
`while` loop	

[condition]
{
action;
}
	[condition]
	{
	action;
	}

Straight Line Flow Chart Pattern	Curved Line Flow Chart Pattern
`do while` loop	

The following patterns are alternately used for If-then-else-if constructs within Stateflow Flowcharts:



Straight Line Flow Chart Pattern	Alternate Straight Line Flow Chart Pattern
`if then else if`	

Straight Line Flow Chart Pattern	Alternate Straight Line Flow Chart Pattern
Cascade of `if then`	
[condition1] { action1; } [condition2] { action2; } [condition3] { action3; }	[condition1] { action1; } [condition2] { action2; } [condition3] { action3; }

Background Information on Basic Blocks and Signals

Basic Blocks

This document uses the term *basic blocks* to refer to blocks built into the Simulink block libraries. The following table lists some examples of basic blocks.

Basic Blocks

Block	Example
Inport	1
Constant	1
Gain	1
Sum	+ +
Switch	
Saturation	
Abs	|u|

Signals and Signal Labels

Signals may be scalars, vectors, or busses. They may carry data or control flows.

You use signal labels to make model functionality more understandable from the Simulink diagram. You can also use them to control the variable names used in simulation and code generation. Enter signal labels only once (at the point of signal origination). Often, you may want to also display the signal name elsewhere in the model. In these cases, the signal name should be inherited until the signal is functionally transformed. (Passing a signal through an integrator is functionally transforming. Passing a signal through an Inport into a nested subsystem is not.) Once a named signal is functionally transformed, associate a new name with it.

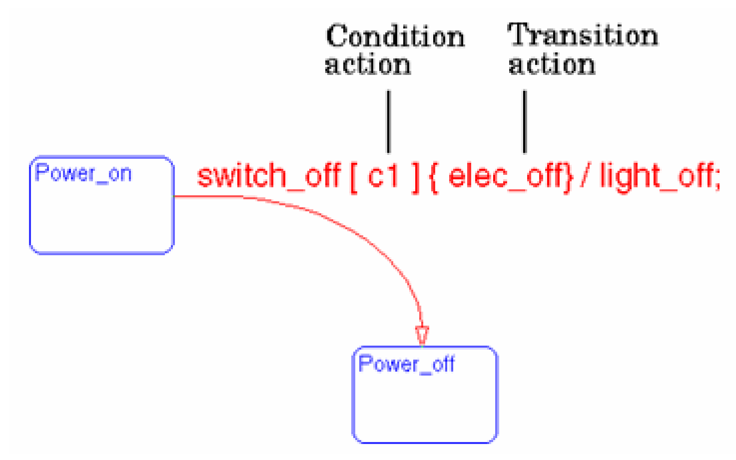
Unless explicitly stated otherwise, the guidelines in "Signals" on page 5-30 apply to all types of signals.

For more information about the representation of signals in Simulink models, see "Working with Signals" in the Simulink documentation.

Glossary

Actions

Actions are part of Stateflow diagram execution. The action can be executed as part of a transition from one state to another, or depending on the activity status of a state. Transitions can have condition actions and transition actions. For example,



States can have `entry`, `during`, `exit`, and, `on event_`*`name`* actions. For example,

```
Power_on/
entry:action1();
during: action2();
exit:action3();
on switch_off:action4();
```

If you enter the name and backslash followed directly by an action or actions (without the entry keyword), the actions are interpreted as entry actions. This shorthand is useful if you are specifying only entry actions.

The action language defines the categories of actions you can specify and their associated notations. An action can be a function call, an event to be broadcast, a variable to be assigned a value, and so on.

Action Language

Sometimes you want actions to take place as part of Stateflow diagram execution. The action can be executed as part of a transition from one state to another, or it can depend on the activity status of a state. Transitions can have condition actions and transition actions. States can have `entry`, `during`, `exit`, and, `on event_name` actions. An action can be a function call, an event to be broadcast, a variable to be assigned a value, etc.

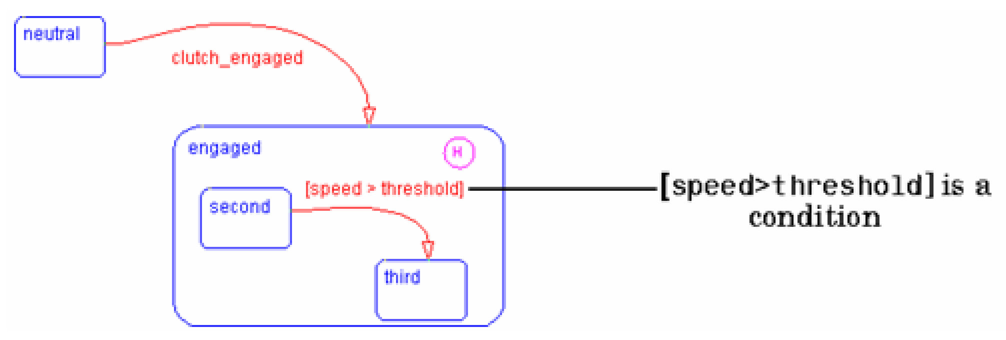
The action language defines the categories of actions you can specify and their associated notations. Violations of the action language notation are flagged as errors by the parser. This section describes the action language notation rules.

Chart Instance

A chart instance is a link from a Stateflow model to a chart stored in a Simulink library. A chart in a library can have many chart instances. Updating the chart in the library automatically updates all the instances of that chart.

Condition

A condition is a Boolean expression to specify that a transition occur, given that the specified expression is true. For example,

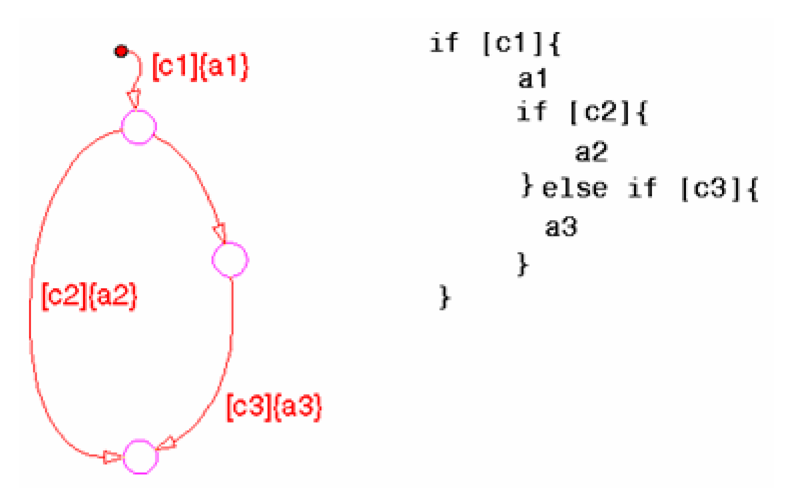


The action language defines the notation to define conditions associated with transitions.

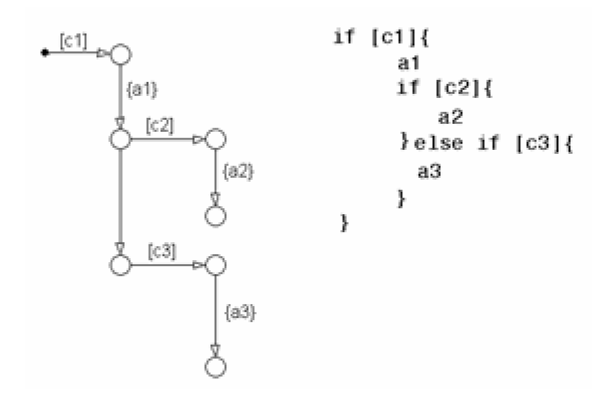
Connective Junction

Connective junctions are decision points in the system. A connective junction is a graphical object that simplifies Stateflow diagram

representations and facilitates generation of efficient code. Connective junctions provide alternative ways to represent the system behavior you want. This example shows how connective junctions (displayed as small circles) are used to represent the flow of an if code structure.



Or the equivalent squared style



Name	Button Icon	Description
Connective junction		One use of a Connective junction is to handle situations where transitions out of one state into two or more states are taken based on the same event but guarded by different conditions.

Data

Data objects store numerical values for reference in the Stateflow diagram.

Defining Data

A state machine can store and retrieve data that resides internally in its own workspace. It can also access data that resides externally in the Simulink model or application that embeds the state machine. When creating a Stateflow model, you must define any internal or external data referenced by the state machine's actions.

Data Dictionary

The data dictionary is a database where Stateflow diagram information is stored. When you create Stateflow diagram objects, the information about those objects is stored in the data dictionary, once you save the Stateflow diagram.

Decomposition

A state has decomposition when it consists of one or more substates. A Stateflow diagram that contains at least one state also has decomposition. Representing hierarchy necessitates some rules around how states can be grouped in the hierarchy. A superstate has either parallel (AND) or exclusive (OR) decomposition. All substates at a particular level in the hierarchy must be of the same decomposition.

Parallel (AND) State Decomposition. Parallel (AND) state decomposition is indicated when states have dashed borders. This representation is appropriate if all states at that same level in the hierarchy are active at the same time. The activity within parallel states is essentially independent.

Exclusive (OR) State Decomposition. Exclusive (OR) state decomposition is represented by states with solid borders. Exclusive (OR) decomposition is used to describe system modes that are mutually exclusive. Only one state, at the same level in the hierarchy, can be active at a time.

Default Transition

Default transitions are primarily used to specify which exclusive (OR) state is to be entered when there is ambiguity among two or more neighboring exclusive (OR) states. For example, default transitions specify which substate of a superstate with exclusive (OR) decomposition the system enters by default in the absence of any other information. Default transitions are also used to specify that a junction should be entered by default. A default transition is represented by selecting the default transition object from the toolbar and then dropping it to attach to a destination object. The default transition object is a transition with a destination but no source object.

Name	Button Icon	Description
Default transition		Use a Default transition to indicate, when entering this level in the hierarchy, which state becomes active by default.

Events

Events drive the Stateflow diagram execution. Define all events that affect the Stateflow diagram. The occurrence of an event causes the status of the states in the Stateflow diagram to be evaluated. The broadcast of an event can trigger a transition to occur and/or can trigger an action to be executed. Events are broadcast in a top-down manner starting from the event's parent in the hierarchy.

Finite State Machine

A finite state machine (FSM) is a representation of an event-driven system. FSMs are also used to describe reactive systems. In an event-driven or reactive system, the system transitions from one mode or state, to another prescribed mode or state, provided that the condition defining the change is true.

Flow Graph

A flow graph is the set of Flowcharts that start from a transition segment that, in turn, starts from a state or a default transition segment.

Flowchart (also known as Flow Path)

A Flowchart is an ordered sequence of transition segments and junctions where each succeeding segment starts on the junction that terminated the previous segment.

Flow Subgraph

A flow subgraph is the set of Flowcharts that start on the same transition segment.

Hierarchy

Using hierarchy you can organize complex systems by placing states within other higher-level states. A hierarchical design usually reduces the number of transitions and produces neat, more manageable diagrams.

History Junction

A History Junction specifies the destination substate of a transition based on historical information. If a superstate has a History Junction, the transition to the destination substate is defined to be the substate that was most recently visited. The History Junction applies to the level of the hierarchy in which it appears.

Name	Button Icon	Description
History Junction	H	Use a History Junction to indicate, when entering this level in the hierarchy, that the last state that was active becomes the next state to be active.

Inner Transitions

An inner transition is a transition that does not exit the source state. Inner transitions are most powerful when defined for superstates with XOR decomposition. Use of inner transitions can greatly simplify a Stateflow diagram.

Library Link

A library link is a link to a chart that is stored in a library model in a Simulink block library.

Library Model

A Stateflow library model is a Stateflow model that is stored in a Simulink library. You can include charts from a library in your model by copying them. When you copy a chart from a library into your model, Stateflow does not physically include the chart in your model. Instead, it creates a link to the library chart. You can create multiple links to a single chart. Each link is called a chart instance. When you include a chart from a library in your model, you also include its state machine. A Stateflow model that includes links to library charts has multiple state machines. When Stateflow simulates a model that includes charts from a library model, it includes all charts from the library model even if there are links to only some of its models. However, when Stateflow generates a stand-alone or Simulink® Coder™ target, it includes only those charts for which there are links. A model that includes links to a library model can be simulated only if all charts in the library model are free of parse and compile errors.

Machine

A machine is the collection of all Stateflow blocks defined by a Simulink model exclusive of chart instances (library links). If a model includes any library links, it also includes the state machines defined by the models from which the links originate.

Nonvirtual Block

Blocks that perform a calculation, such as a Gain block.

Notation

A notation defines a set of objects and the rules that govern the relationships between those objects. Stateflow notation provides a common language to communicate the design information conveyed by a Stateflow diagram. Stateflow notation consists of:

- A set of graphical objects
- A set of nongraphical text-based objects
- Defined relationships between those objects

Parallelism

A system with parallelism can have two or more states that can be active at the same time. The activity of parallel states is independent. Parallelism is represented with a parallel (AND) state decomposition.

Real-Time System

A system that uses actual hardware to implement algorithms, for example, digital signal processing or control applications.

Simulink Coder

Simulink Coder software includes an automatic C language code generator for Simulink. It produces C code directly from Simulink block diagram models and automatically builds programs that can be run in real-time in a variety of environments.

Simulink Coder Target

An executable built from code generated by the Simulink Coder product.

S-function

A customized Simulink block written in C or MATLAB-code. S-functions written in C can be inlined in the Simulink Coder software. When using Simulink together with Stateflow for simulation, Stateflow generates an S-function (MEX-file) for each Stateflow machine to support model simulation. This generated code is a simulation target and is called the S-Fun target within Stateflow.

Signal propagation

Process used by Simulink to determine attributes of signals and blocks, such as data types, labels, sample time, dimensionality, and so on, that are determined by connectivity.

Signal source

The signal source is the block of origin for a signal. The signal source may or may not be the true source.

Simulink

Simulink is a software package for modeling, simulating, and analyzing dynamic systems. It supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two. Systems can

also be multirate, that is, have different parts that are sampled or updated at different rates.

Simulink allows you to represent systems as block diagrams that you build using your mouse to connect blocks and your keyboard to edit block parameters. Stateflow is part of this environment. The Stateflow block is a masked Simulink model. Stateflow builds an S-function that corresponds to each Stateflow machine. This S-function is the agent Simulink interacts with for simulation and analysis.

The control behavior that Stateflow models complements the algorithmic behavior modeled in Simulink block diagrams. By incorporating Stateflow diagrams into Simulink models, you can add event-driven behavior to Simulink simulations. You create models that represent both data and control flow by combining Stateflow blocks with the standard Simulink blockset. These combined models are simulated using Simulink.

State

A state describes a mode of a reactive system. A reactive system has many possible states. States in a Stateflow diagram represent these modes. The activity or inactivity of the states dynamically changes based on events and conditions.

Every state has hierarchy. In a Stateflow diagram consisting of a single state, that state's parent is the Stateflow diagram itself. A state also has history that applies to its level of hierarchy in the Stateflow diagram. States can have actions that are executed in a sequence based upon action type. The action types are: `entry`, `during`, `exit`, or `on event_name` actions.

Name	Button Icon	Description
State		Use a state to depict a mode of the system.

Stateflow Block

The Stateflow block is a masked Simulink model and is equivalent to an empty, untitled Stateflow diagram. Use the Stateflow block to include a Stateflow diagram in a Simulink model.

The control behavior that Stateflow models complements the algorithmic behavior modeled in Simulink block diagrams. By incorporating Stateflow blocks into Simulink models, you can add complex event-driven behavior to Simulink simulations. You create models that represent both data and control flow by combining Stateflow blocks with the standard Simulink and toolbox block libraries. These combined models are simulated using Simulink.

Stateflow Debugger

Use the Stateflow Debugger to debug and animate your Stateflow diagrams. Each state in the Stateflow diagram simulation is evaluated for overall code coverage. This coverage analysis is done automatically when the target is compiled and built with the debug options. The Debugger can also be used to perform dynamic checking. The Debugger operates on the Stateflow machine.

Stateflow Diagram

Using Stateflow, you create Stateflow diagrams. A Stateflow diagram is also a graphical representation of a finite state machine where states and transitions form the basic building blocks of the system.

Stateflow Explorer

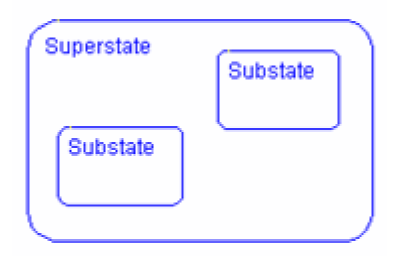
Use the Stateflow Explorer to add, remove, and modify data, event, and target objects.

Stateflow Finder

Use the Finder to display a list of objects based on search criteria that you specify. You can directly access the properties dialog box of any object in the search output display by clicking on that object.

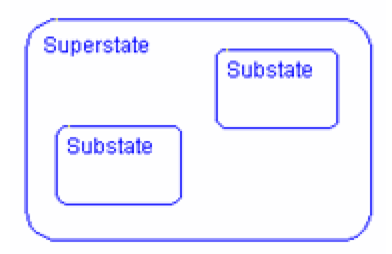
Substate

A state is a substate if it is contained by a superstate.



Superstate

A state is a superstate if it contains other states, called substates.



Target

An executable program built from code generated by Stateflow or Simulink Coder software.

Top-down Processing

Top-down processing refers to the way in which Stateflow processes states. In particular, Stateflow processes superstates before states. Stateflow processes a state only if its superstate is activated first.

Transition

A transition describes the circumstances under which the system moves from one state to another. Either end of a transition can be attached to a source and a destination object. The source is where the transition begins and the destination is where the transition ends. It is often the occurrence of some event that causes a transition to take place.

Transition Path

A transition path is a Flowchart that starts and ends on a state.

Transition Segment

A transition segment is a single directed edge on a Stateflow diagram. Transition segments are sometimes loosely referred to as transitions.

Tunable parameters

A tunable parameter is a parameter that can be adjusted in the model and in generated code.

True Source

The true source is the block which creates a signal. The true source is different from the signal source because the signal source may be a simple routing block such as a Demux block.

Virtual Block

When creating models, be aware that Simulink blocks fall into two basic categories: nonvirtual and virtual blocks. Nonvirtual blocks play an active role in the simulation of a system. If you add or remove a nonvirtual block, you change the model's behavior. Virtual blocks, by contrast, play no active role in the simulation. They help to organize a model graphically. Some Simulink blocks can be virtual in some circumstances and nonvirtual in others. Such blocks are called conditionally virtual blocks. The following table lists Simulinks virtual and conditionally virtual blocks.

Block Name	Condition Under Which Block Is Virtual
Bus Selector	Virtual if input bus is virtual
Demux	Always virtual
Enable	Virtual unless connected directly to an Outport block
From	Always virtual
Goto	Always virtual
Goto Tag Visibility	Always virtual
Ground	Always virtual
Inport	Virtual when the block resides within any subsystem block (conditional or not), and does not reside in the root (top-level) Simulink window.

Block Name	Condition Under Which Block Is Virtual
Mux	Always virtual
Outport	Virtual when the block resides within any subsystem block (conditional or not), and does not reside in the root (top-level) Simulink window.
Selector	Virtual except in matrix mode
Signal Specification	Always virtual
Subsystem	Virtual unless the block is conditionally executed and/or the block's Treat as Atomic Unit option is selected.
Terminator	Always virtual
Trigger	Virtual if the Outport port is not present.

Virtual Scrollbar

Using a virtual scrollbar, you can set a value by scrolling through a list of choices. When you move the mouse over a menu item with a virtual scrollbar, the cursor changes to a line with a double arrowhead. Virtual scrollbars are either vertical or horizontal. The direction is indicated by the positioning of the arrowheads. Drag the mouse either horizontally or vertically to change the value.